Proceedings of the IEEE Real-Time Systems Symposium (RTSS 2004), Lisbon, Portugal, December 2004.

Real-Time Resource Reservation Protocol for Wireless Mobile Ad Hoc Networks

Tullio Facchinetti
University of Pavia, Italy
tullio.facchinetti@unipv.it

Luis Almeida
University of Aveiro, Portugal
lda@det.ua.pt

Giorgio C. Buttazzo
University of Pavia, Italy
buttazzo@unipv.it

Carlo Marchini
University of Parma, Italy
carlo.marchini@unipr.it

Abstract

Wireless communication technology is spreading quickly in almost all the information technology areas as a consequence of a gradual enhancement in quality and security of the communication, together with a decrease in the related costs. This facilitates the development of relatively low-cost teams of autonomous (robotic) mobile units that cooperate to achieve a common goal. Providing real-time communication among the team units is highly desirable for guaranteeing a predictable behavior while operating autonomously in unstructured environments.

This paper proposes a MAC protocol for wireless communication that supports dynamic resource reservation for small teams of cooperative robots. The protocol uses a slotted time-triggered medium access transmission control that is collision-free, even in the presence of hidden nodes. The transmissions are scheduled according to the Earliest Deadline First scheduling policy. An adequate admission control guarantees the timing constraints of the team communication requirements, including when new nodes dynamically join or leave the team. The paper describes the protocol focusing on the consensus procedure that supports coherent changes in the global system. Finally, a set of simulation results are shown that illustrate the effectiveness of the proposed protocol.

1 Introduction

Teams of autonomous mobile robots may represent an attractive solution for situations in which the environment conditions are not suitable for direct human intervention. Sample application domains include space missions, hazardous environment exploration, demining, surveillance, and civil protection [6]. In these cases, small teams of robots are required to operate autonomously in open environments for monitoring and exploration purposes. In addition, they have to cooperate for achieving a common goal. Communication systems based on wired backbones are not usually suitable for this kind of applications because it is often impossible to deploy a wired infrastructure in open or remote spaces. As a consequence, a full autonomy of the robotic team can only be achieved through a wireless ad-hoc network [14].

Achieving real-time communication over wireless networks has always been a challenge [12] [2] due, mainly, to the higher attenuation and higher bit error rates typical of that medium. The challenge is, however, substantially larger when the nodes move and establish ad-hoc links as in wireless mobile ad-hoc networks (MANETs) [7]. It is interesting to notice that these networks differ from sensor networks [12] in at least two ways: they are not always large scale, which means scalability might not be an issue, and physical constraints are not as stringent, which means that more powerful processors, radio transceivers and batteries can generally be used. This latter aspect does not mean, however, that resource-consciousness is not an issue. It still is but generally at a lower importance than in sensor networks. On the other hand, MANETs differ from industrial wireless networks [2] because the latter ones are frequently structured, i.e. based on fixed access points.

A further challenge in MANETs is supporting dynamic resource reservation as required by nodes that join or leave the network at run-time, or by changes in the communication requirements. This is necessary for an efficient use of the communication bandwidth and for flexibility with respect to the operational environment.

This paper proposes a communication protocol for MANETs targeted to small teams of mobile autonomous robots that move in the vicinity of each other. The protocol supports dynamic resource management with adequate admission control, thus respecting the communication timing constraints, even in the presence of communication errors and hidden nodes. To support dynamic resource management the protocol uses a consensus procedure that allows all nodes to be aware of the changes in resource allocation. This procedure is the main focus of this paper.

The paper is organized as follows: Section 2 presents a brief survey of related work and Section 3 introduces the

system model. Then, Section 4 describes the consensus procedure while Section 5 presents and validates an upper bound on the time taken by the consensus procedure. Section 6 includes simulations results that show the effectiveness of the protocol even with errors and mobility. Finally, Section 7 states our conclusions and future work.

2 Related work

Wireless communication technology has recently become pervasive in many application domains, enabled by a gradual enhancement in quality and security of the communication, together with a substantial decrease in the related costs. The resulting wireless networks are normally classified in two categories: structured, i.e., based on fixed access points; and ad-hoc. A further classification divides the latter category into mobile ad-hoc networks (MANETs) and sensor networks [12].

All categories have been extensively addressed by the research community but only a relatively small subset of the vast amount of available literature addresses aspects related with real-time communication. Some fundamental aspects that constrain the real-time behaviour are the medium access control (MAC) protocol, the routing mechanisms and the mechanisms to handle dynamic communication requirements. This paper deals with these aspects in the scope of MANETs, particularly for small teams of autonomous mobile robots, i.e., with around 10 units. The approaches herein proposed can still be used for larger numbers of units but within an adequate hierarchical framework.

One of the aspects that makes obtaining real-time behavior in MANETs particularly challenging is mobility. In fact, as nodes move, the links between nodes may break and new links may be established, leading to a dynamic topology. To deal with mobility, MANETs typically use specific techniques. For example, in [5], the link duration for different mobility scenarios is analyzed in order to deduce adaptive metrics to identify more stable links. Another possible approach is to manage the network topology by controlling the positioning of certain or all nodes. This is proposed in [11], where a set of specific nodes (PILOT nodes) is oriented toward specific places to support the connectivity of the remaining nodes (general sensor nodes) in order to sustain real-time communication. The issue of real-time communication and mobility is analyzed in [7], where mobility awareness and prediction are proposed to perform proactive routing and resource reservation to allow meeting real-time constraints. However, they do not propose a specific algorithm or method to achieve this.

The option behind our work is also of using a flat proactive routing algorithm for maintaining information about the current topology as well as for performing changes on global information as required for resource reservation. But we propose a specific algorithm based on a variation of flooding, with each node broadcasting periodically its topology estimation together with information for synchronization purposes. Then each node merges the received information with its own current local view and uses this updated information in its next synchronization broadcast. One of the main results of this paper is the determination of an upper bound on the number of steps, i.e., synchronization broadcasts, required to assure the propagation of new information through all the network. This bound is particularly relevant for the consensus procedure that supports resource reservation.

The problem of reaching a consensus has been widely considered in the literature on distributed systems since it was firstly introduced in [10]. Dolev *et al.* [3] proved that in a system with clock synchronization and time-bounded communications, such as ours, it is possible to reach a consensus. The consensus procedure proposed in this paper is optimistic in the sense that, upon a change request, an instant in time is defined into the future, at which the procedure should be concluded. At that instant, nodes check an aggregated positive acknowledgement, which was disseminated through the network after the request, and determine whether there was an agreement among all nodes. The change request is executed only in case of consensus. In this paper, we will use the expressions *consensus* and *agreement* interchangeably.

Finally, as far as the MAC protocol is concerned, this paper proposes the use of implicit EDF [1], which was originally designed for use within static cells of hierarchical sensor networks. We combine implicit EDF with the referred consensus procedure to support dynamic communication requirements and, generally, dynamic resource reservation. A preliminary approach to such combination was first proposed in [4] but with the restrictive assumption of absence of hidden nodes, a restriction that is now lifted.

3 System model

System architecture: The global system architecture considered in this paper consists of a set Π of n_π mobile units or nodes, $\Pi = \{p_1, \ldots, p_{n_\pi}\}$, which can communicate over a radio-based wireless medium. Every unit is unambiguously identified by a statically-assigned identifier $Id(p_i) = Id_i$. All the nodes use a single shared radio channel to exchange messages. The nodes are not location-aware and the topology is not managed meaning that there is no topology-oriented control of the nodes movement.

We say that there exists a link L_{ij} from node p_i to node p_j if p_j is able to listen to a transmission from p_i. Such a link is represented by the edge $p_i \rightarrow p_j$ in the topology graph. A set of links connecting two nodes p_i and p_j establishes a path between them. A path from p_i to p_j will be denoted as $p_i \equiv p_{m_1} \rightarrow \ldots \rightarrow p_{m_{s-1}} \rightarrow p_{m_s} \equiv p_j$. Then, a team (or network) $\pi(t) \subseteq \Pi$ is defined as a dynamic subset of $n(t)$ nodes from Π, $\pi(t) = \{p_1, \ldots, p_{n(t)}\}$. If not explicitly declared, in the following sections we will refer unambiguously to $n(t)$ as n and to $\pi(t)$ as π. A team is fully connected if for any pair of nodes $p_i, p_j \in \pi(t)$ there exists at least a path between them. More restrictively, a team is fully linked if for any pair of nodes $p_i, p_j \in \pi(t)$

there exists a link between them.

In order to maintain topological information of the network at each instant, each node p_k uses a topology matrix M^k, with $n \times n$ elements, which can be considered as the adjacency matrix for an oriented graph. The generic element M^k_{ij} placed in the i-th row and j-th column is a flag indicating what node p_k knows about the link L_{ij}. We set $M^k_{ij} = 1$ ($i \neq j$) if there exists such a link and $M^k_{ij} = 0$ ($i \neq j$) otherwise; we set $M^k_{ii} = 0$ for each i by default. The M^k matrix is dynamic since the units are moving, thus it changes over time as new links are established or broken. Therefore, we will use $M^k(t)$ to refer to the topology matrix owned by node p_k at instant t.

Each node updates its own topology matrix whenever it receives a message, whenever it does not receive an expected message, or when it receives a matrix of a neighbour node, which are broadcast periodically. The algorithm that rules the topology matrix update is fully distributed and converges to a coherent global view. However, it is not described in this paper due to space constraints. Moreover, it is relatively marginal to the consensus process described in this paper since it is currently used for detection of crashes or lost nodes, only, in which case the respective column will be null. However, the topology matrix will, in future work, support a variety of functions, e.g. topology management, routing, spatial reuse of the communication channel and determination of tighter bounds on the duration of the consensus procedure.

Finally, some of the results presented later on are based on the so-called *level of redundancy* R of a given topology, which is defined as the ratio between the actual number of links of that topology over the maximum number of links for the same number of nodes. Both terms of the ratio only count the links beyond the minimum, i.e., those that are essential to keep the network fully connected. R varies between 0 and 1 and gives an indication of the number of redundant paths that a given topology contains.

Communication model: Communication among nodes is organized in consecutive slots, which have a constant duration T_{slot}. The model is periodic, which means that all message streams served by the communication system are periodic, that is, made of a potentially infinite sequence of message instances submitted periodically for transmission. For the sake of simplicity, the expression *message* will also be used to refer to a *message stream*, unless stated otherwise. Message addressing is content-based, making use of an identifier. Furthermore, the communication follows a producer-consumer model, according to which producers broadcast their messages autonomously, with a given frequency, while consumers retrieve from the network the messages that are relevant for them.

The generic message m_l generated by node p_i is characterized by its identifier I_l, a transmission period T_l, a relative deadline D_l, an offset O_l, and a transmission duration C_l, all (except the identifier) expressed in *slots*. The Communication Requirements Table (CRT) holds the properties of all the messages to be scheduled by the communication system, so $CRT = \{m_l(I_l, C_l, T_l, D_l, O_l), l = 1 \ldots N\}$, where N is the number of message streams produced by all nodes. The total bandwidth requirement is given by $U_{CRT} = \sum_{l=1}^{N} \frac{C_l}{T_l}$.

As referred in Section 2, messages are scheduled using the implicit EDF approach [1]. Each message is transmitted as a sequence of fixed size packets, each of which is transmitted in a single slot. Implicit EDF considers that message preemption is possible at the slot boundaries, i.e., between packets. Since all messages also become ready for transmission synchronously with the slot boundary, then, this scheduling model is equivalent to preemptive EDF [9].

We say that the traffic model is dynamic since existing team nodes may request changes in their message streams, or nodes not in the team may request to join, or even nodes in the team may request to leave or just crash. In all these circumstances, the CRT must be updated. Since the CRT is replicated in all the nodes together with the EDF scheduler, a consensus process is required to reach an agreement among all nodes in the team concerning the CRT update, including hidden nodes. Whenever it is necessary to refer to each CRT replica separately, we will use $CRT^k(t)$ meaning the replica within node p_k at instant t.

To support topology self-checking, synchronization, and admission control, each node p_k periodically broadcasts a message with its own $CRT^k(t)$, $M^k(t)$, local clock value $clk_k(t)$ and other information related with the consensus procedure triggered upon CRT change requests. This is called the system synchronization message m_{sync} and it is broadcasted by all nodes in a round-robin fashion ($p_k, \ldots, p_{n-1}, p_n, p_1, \ldots, p_{k-1}$). We will call *step* the transmission of a synchronization message. The ensemble of all these messages constitutes a periodic message stream with period T_{sync}, called the *synchronization step period*, and duration C_{sync}. However, each instance of this message stream is transmitted by a different node according to the round-robin sequence based on the node identifier. Figure 1 shows an example of schedule of the communication activity, with 3 nodes sending one message each, plus the synchronization message. In that case, each message uses a single slot only, i.e., $C_{1\ldots3} = C_{sync} = 1$, and the step period is 5, i.e., $T_{sync} = 5$.

From a traffic scheduling point of view, m_{sync} is like another periodic message, scheduled together with the remaining messages by the implicit EDF scheduler, with period T_{sync}, deadline $D_{sync} = T_{sync}$, offset $O_{sync} = 0$ and duration C_{sync}. Each node knows when to transmit its own m_{sync} by checking the round-robin list and sends the m_{sync} message once every *synchronization round*, with period $T_{round} = nT_{sync}$.

The total bandwidth consumed by our communication system is given by $U_{tot} = \sum_{i=1}^{N} \frac{C_i}{T_i} + \frac{C_{sync}}{T_{sync}}$ and a sufficient and necessary traffic schedulability condition can be obtained by $U_{tot} \leq 1$. Notice that U_{tot} includes all overheads, such as all the control information sent in each slot,

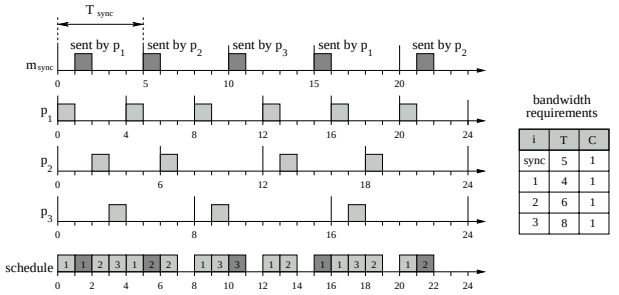

Figure 1. Example showing the m_{sync} message broadcast.

as well as any unused space within the slots.

Finally, the clock sent within the synchronization message ($clk_i(t)$) includes both a representation of continuous time (i.e, with microseconds resolution) and an absolute $slot$ counter (slot counter). The former is used for clock synchronization purposes, while the latter is used for scheduling and consensus purposes. For clarity of presentation, we will use $clk_i(t)$ to refer to the $slot$ counter, only, unless explicitly stated otherwise.

Fault model: The radio wireless medium is very error prone due, for example, to electromagnetic interferences, signal fading, multiple transmission paths and broadcasting collisions. Therefore, we assume that a receiving node can check the integrity of a message detecting the presence of errors in the broadcast, e.g. using CRC checking. Malicious faults (byzanthine faults [8]) are not considered in this fault model, meaning that we assume that the content of a valid message is reliable and it is not intentionally corrupted by the sending node.

Since the MAC protocol is time-triggered, it is sensitive to clock drifts and clock errors. These will result in collisions and are thus considered as another source of communication errors. Moreover, the slot duration does not account for retransmissions and thus errors will be handled in a forward error recovery fashion, only. In particular, we consider that erroneous messages lead to omissions.

The errors affecting data messages will necessarily cause a degradation of the quality-of-service delivered to the application, which must deal with such situations. In this paper we are concerned with errors affecting synchronization messages, only, because omissions of these messages impact directly on the operation of the communication system.

As briefly explained in section 2, the protocol proposed in this paper relies on the precision of clock synchronization and on the effectiveness of the consensus procedure. The impact of errors on these mechanisms depends heavily on the level of redundancy R of the current topology as shown further on in section 6. In fact, low values of R correspond to topologies in which there are few or none alternative paths between the two most distant nodes and thus omissions may impose extra delays in forwarding synchronization messages. On the other hand, high values of R correspond to topologies in which there are many parallel paths between any pair of nodes leading to a high resilience to omissions of those messages, since they are sent in a flooding fashion.

Finally, we also account for possible crashes or nodes that lost contact with the team. The recovery mechanism from such cases makes use of an automatic startup procedure that, however, causes a temporary disruption of the communication. The currently devised startup procedure is based on a special node, the team leader, that, upon a timeout without detecting any on-going communication activity starts transmitting its synchronization message, allowing other waiting nodes to join, one by one, building up the team.

4 Reaching a consensus

Whenever a global decision must be taken by the team, for example concerning a change in the communication schedule triggered by a joining request from a new robot or a request for changes in the bandwidth requirements, it is important to guarantee that such decision is consistent for all the members and that it is taken at the same time, because the schedule is computed independently on each node. This is achieved by keeping track of the knowledge the other team units have about the decision to take. Such a knowledge is stored in a data structure, called the *agreement vector* A, which is broadcasted by all nodes within the synchronization message. The agreement vector is an array of n elements, owned by each member of the team, where A^k denotes the vector owned by node p_k. The i-th element A_i^k of the vector is a binary flag indicating whether node p_i has been notified of the global decision. When marked ($A_i^k = 1$), it means that node p_k knows that node p_i is aware of the decision. Therefore, A represents an aggregated acknowledge of the global awareness of the decision to be taken at a defined time in the future.

4.1 The consensus process

In the field of distributed systems there is a substantial amount of work in consensus processes. These must generally enforce the following properties [13]: Termination, Validity and Agreement. Below, we state these properties in the scope of our consensus model, which presents some specific features that are different from traditional ones:

1. *Termination:* The consensus process stops anyway at a given time t, whether or not the agreement has been reached. This is explicitly enforced by our protocol by setting a termination time *a priori*, when a consensus process is triggered.
2. *Validity:* Any consensus process is meaningful in the sense that it is triggered by the system for the sake of the system correct operation. This property is enforced by our fault model because it does not consider malicious faults, such as those in which an erroneous process could be triggered or a node could purposely jeopardize an on-going process.

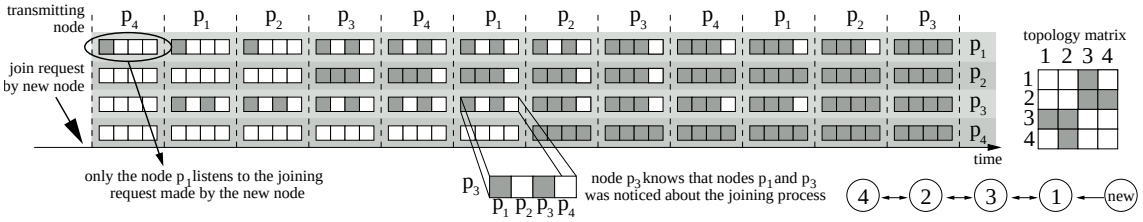

Figure 2. Example of the agreement vector update.

3. *Agreement:* At the process termination time t, two or more nodes can have different information concerning the consensus process status and thus, decide differently. However, such an inconsistency does not jeopardize the consistent operation of the system. This is enforced by a positive aggregated acknowledge of the consensus process in all nodes that allows to differentiate those that reached consensus, which will follow on, from those that did not, which will stop and resynchronize with the former ones. Such an aggregated acknowledge is based on the agreement vector A.

4.2 Triggering a new process

For a node p_k to trigger a consensus process it must:

1. Assign a unique identifier $proc_k$ to the process. Notice that the round-robin circulation of the synchronization message transmission ensures that only one node can trigger an agreement process at any given time. Therefore, each process can be uniquely identified by the clock value at the time it will be triggered, i.e., $proc_k = clk_k(t)$. Recall that $clk_k(t)$ is the $slot$ counter value of the slot in which m_{sync} is sent.
2. Wait for its turn to broadcast the synchronization message m_{sync}.
3. If there is another process already running in the system, the vector A^k owned by p_k is not empty. In that case, p_k cannot start a new process, which must be retriggered later.
4. Otherwise, or after the termination of the previous process, mark the cell A^k_k in an empty (new) vector.
5. Associate to the consensus process the identifier Id_i of the node that issued the request (possibly, $i = k$). This is necessary to differentiate between several requests that can arrive to the same node p_k, before it can trigger the respective processes (e.g. p_6 in Figure 3 can receive requests from p_{new2} and p_{new3}).
6. Set the agreement time t_a equal to the triggering time $clk_k(t)$ plus an upper bound on the duration of the consensus process, as derived further on ($S(n)T_{sync}$). The agreement time t_a is the time at which all nodes will simultaneously update the communication system data, including the CRT, matrix M, vector A, and the round-robin circulation list.
7. Send the synchronous message m_{sync} with the updated agreement information, i.e. $proc_k$, Id_i, A, t_a.

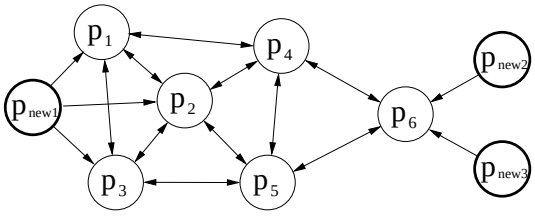

Figure 3. Example of simultaneous starts of multiple consensus processes.

To enforce data consistency during a consensus process, it is crucial that n does not change in the middle of the process (otherwise, it could invalidate the update instant, for example). This is achieved by preventing a node from triggering a new consensus process when there is an on-going one, as stated in the rules above. However, since the processes take time to propagate, it is possible that one node triggers a process without knowing that another process is already in progress. For example, in Figure 3, node p_6 could trigger one consensus process to admit p_{new2}, while p_1 could trigger another one in the following cycle to admit p_{new1}. As both processes propagate, there must be at least one node in their paths that receives both consensus processes. When this happens, one of the processes is allowed to progress until completion while the other is dropped and must be re-issued later.

4.3 Updating the agreement vector

When node p_k receives an agreement vector from another node, p_w, several situations can occur:

1. If node p_k is not currently engaged in any consensus process, i.e., A^k is empty, it performs the following operations:
   (a) $A^k_k = 1$
   (b) $A^k = A^k | A^w$ (symbol $|$ denotes the bitwise OR operator).
2. If node p_k is currently engaged in one on-going agreement process, i.e., A^k is not empty, then it must check whether the received vector corresponds to the same process or a different one.
   (a) If $proc_k = proc_w$, then it is the same process and thus p_k updates its vector with the received one: $A^k = A^k | A^w$.
   (b) If $proc_k < proc_w$, the process corresponding to A^k is older than the one in A^w, thus A^w is discarded while A^k is kept unchanged.

(c) If $proc_k > proc_w$, the process corresponding to A^k is newer than the one in A^w, thus A^k is replaced by A^w while its previous contents are discarded. Moreover, the self flag is marked, i.e., $A^k_k = 1$.

The bitwise OR operation in rules 1b and 2a captures the knowledge that node p_w has about the nodes that were already notified of the consensus process, and passes that knowledge to p_k.

Rules 1a and 2c refer to situations in which p_k is notified about the consensus process, marking its own flag in the vector.

In rules 2b and 2c an on-going process is discarded. The requester of this process will be indirectly informed of this situation since it will eventually receive an m_{sync} message containing a different consensus process. The requester must then re-issue the request at a time after the agreement time of the on-going consensus process.

4.4 Termination of a consensus process

As referred in Section 4.2, the termination instant of any consensus process t_a is set at the time the process is triggered and it is disseminated through all the network. If the network is fully connected, in the absence of errors, broken links and crashes or absent nodes, it is possible to prove (see Section 5) that at time t_a the process will be *complete*, for any network topology.

Definition 1 *Given a node $p_i \in \pi(t)$ and its corresponding agreement vector A^i, the consensus process is said to be complete when $\forall i, j = 1, \ldots, n\ A^i_j = 1$.*

The definition above means that all nodes know that a consensus has successfully been reached by all. Therefore, the agreement property is respected and the request relative to the consensus process is executed. However, in reality, both errors, broken links and even crashes can occur. Therefore, it is possible that at instant t_a the consensus process is not *complete* and two situations can happen.

First, consider the case in which the consensus process reached all nodes but some of them have not been notified of that. This means that some nodes have the A vector fully marked while others still have a few unmarked flags. In this case we say the consensus process is *partially complete*:

Definition 2 *Given a node $p_i \in \pi(t)$ and its corresponding agreement vector A^i, the consensus process is said to be partially complete if $\exists i : \forall j = 1, \ldots, n\ A^i_j = 1$.*

Notice that this is still a coherent situation, despite some nodes not knowing it. Therefore, those that reached the consensus, i.e., have a fully marked A vector, execute the request relative to the consensus process. On the other hand, those that did not reach consensus, refrain from transmitting until they receive an m_{sync} message. At that time, they update their own CRT with the one received in m_{sync}, which is properly updated with the previous consensus process, and restart transmitting. This is illustrated in Figure 4, case a). Node p_2 reaches the consensus and starts the new schedule, while nodes p_1 and p_5 stop transmitting to avoid collisions and restart later, after receiving the right CRT from node p_2. Case b) of Figure 4 illustrates an impossible situation because, if node p_5 holds an empty A vector, then the 5-th column of A^1 and A^2 must be unmarked and thus no nodes reach the consensus. This leads to another situation in which the consensus process is *incomplete*.

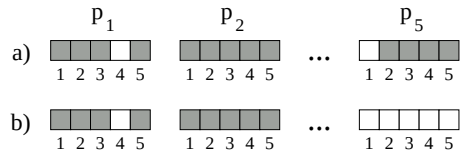

Figure 4. Example of errors in the vector broadcasting.

Definition 3 *Given a node $p_i \in \pi(t)$ and its corresponding agreement vector A^i, the consensus process is said to be incomplete if $\forall i, \exists j = 1, \ldots, n : A^i_j = 0$.*

This situation may occur when a node crashes or departs from the team without being notified of the consensus process, or even in the presence of too many errors. This causes all the nodes in the team to stop transmitting leading to a major communication disruption. To recover from this situation there is a timeout that limits the maximum time that a node waits for an m_{sync} message, after which the node initiates a startup procedure using the previous state of the CRT, i.e., without executing the request.

After restart, however, it will not be possible to reach any other agreement until the crashed or absent node is removed from the team. This can be carried out by using the topology matrix M described in Section 3. In fact, a crashed or absent node is reflected in the topology matrix by an empty column in the respective index. Any node detecting such empty column within M, for a given predefined time, triggers the removal process.

Notice that a consensus process to remove such node(s) is still possible because it will not require their agreement and the respective consensus process does not take into account the respective flags in vector A.

4.5 Adding nodes to the team

In the course of team operation, it may happen that a new node appears and requests to join the team. This action is triggered by the new node, which is outside the team and thus not included in the current communication schedule. Therefore, a special mechanism is required in this case.

An external node that wants to join the team must first listen to the system, scanning for synchronization messages. Upon reception of such a message, sent by node p_k, the first task to be accomplished is to synchronize its clock using clk_k and secondly to examine CRT^k. By inspecting this

table, the joining node executes an admission control to verify whether its communication requirements can be met by the system, given the actual communication load. Upon a positive admission control, the joining node builds the same schedule, as all the team nodes, and indicates its presence by issuing a communication request in a free scheduling slot, submitting its bandwidth requirements to the team members that are within its range of transmission. Following this request, the joining node remains listening, waiting for the synchronization message that carries its request, which is used as an acknowledgment that the respective consensus process has started. If the following m_{sync} does not refer to the issued request, the joining node waits until t_a indicated in that m_{sync}. Then, it further waits for a random number of synchronization cycles to reduce collisions with other possible joining nodes, and re-issues the request. Possible request duplicates received by neighbour team nodes may generate parallel consensus processes, but only the oldest is kept, as discussed in Section 4.3.

5 Validation of the model

In this section we present several results concerning the time taken by the consensus process in the absence of errors, message losses and crashes or absent nodes. Moreover, we will consider that the topology remains fixed for the duration of the consensus process. Then, at the end of this section we present simulation results that show the performance of the protocol when those assumptions do not hold. First, we introduce the following definition:

Definition 4 *The consensus process is said to have converged if it is completed in a finite number of steps.*

Lemma 1 *Given two nodes $p_k, p_w \in \pi$, if there exists at least a path from p_k to p_w, then the information contained in A^k sent by node p_k will be received by p_w after a finite number of steps.*

Proof. When a node receives a non-empty agreement vector from another node, it updates its own agreement vector by marking the flags that are marked in the received vector (updating rule 1b). In this way, the vector forwarded by that node will contain at least the marked flags that were already marked in the received vector. Since every node transmits once in each synchronization round, then there will be a node that forwards the contents of A^k in each round. Since there exists a path from p_k to p_w, such data will be received by p_w and, since the number of nodes in π is finite, then the information is forwarded from p_k to p_w in a finite number of steps. □

Theorem 1 *If for each $p_k \in \pi$ there exists at least one path that starts from p_k and crosses all the nodes in π, then the consensus process converges.*

Proof. From the existence of a path from p_k to all the other nodes, we know that any marked flag in the agreement vector A^k, broadcast by p_k, will be received by every generic node $p_i \in \pi$. Moreover, from Lemma 1, we know that it will be received by p_i in a finite number of steps. When p_i receives such flags of A^k, it marks them within its own vector A^i (updating rule 1b) and marks its self flag A^i_i (updating rule 1a). Similarly, all marked flags of A^i will be received by all the other nodes and also in a finite number of steps. Since this holds for all k or i, the process can be completed (in the sense of Definition 1) in a finite number of steps, which proves the theorem. □

To respect the termination requirement of our consensus model, an estimation of the number of steps needed to complete a consensus process must be supplied. Theorem 2 gives an upper bound of such number of steps for a given topology. It can be used only when the network topology is known. Later in this section we introduce an upper bound that holds for the most unfavourable topology, referred to as worst-case topology, and thus it holds equally for any possible linked topology. We firstly introduce the following definition:

Definition 5 *Given two nodes $p_k, p_w \in \pi$, the step distance $\Delta_s(p_k, p_w)$ between p_k and p_w is defined as*

$$\Delta_s(p_k, p_w) = \begin{cases} w - k & \text{if } k \leq w, \\ n + w - k & \text{if } k > w. \end{cases}$$

The step distance introduced in Definition 5 gives the number of steps (i.e., synchronization periods, T_{sync}) required to have p_w transmitting the m_{sync} message after the time at which p_k transmitted it (round-robin order).

Lemma 2 $\forall i, j (1 \leq i, j \leq n \wedge i \neq j), \Delta_s(p_i, p_j) + \Delta_s(p_j, p_i) = n.$

Proof. The proof follows directly from Definition 5. □

Lemma 3 $\forall k, w = 1, \ldots, n \quad \Delta_s(p_k, p_w) \leq n - 1.$

Proof. If $k = w$ then $\Delta_s(p_k, p_w) = 0 \leq n$. If $k \neq w$ then $\Delta_s(p_w, p_k) \geq 1$ and $\Delta_s(p_k, p_w) \leq n - 1$, from Lemma 2. □

Definition 6 *Let $p_k \equiv p_{m_1} \rightarrow \ldots \rightarrow p_{m_{s-1}} \rightarrow p_{m_s} \equiv p_w$ be a path from p_k to p_w. The following distances are defined:*

$$\Delta_{hop}(p_k, p_w) = s - 1,$$

$$\Delta_t(p_k, p_w) = \sum_{i=1}^{s-1} \Delta_s(p_{m_i}, p_{m_{i+1}}).$$

The distance $\Delta_{hop}(p_k, p_w)$ denotes the number of hops required to transmit a piece of information from p_k to p_w. The distance $\Delta_t(p_k, p_w)$ specifies the number of steps required to have p_w transmitting after it received an information that was initially sent by p_k.

Definition 7 *Let π be a network with a topology matrix M. We say that $d(\pi, M)$ is the maximal distance (or $diameter$) in the network between two nodes if and only if $\forall i, j = 1, \ldots, n \quad \Delta_{hop}(p_i, p_j) \leq d(\pi, M)$.*

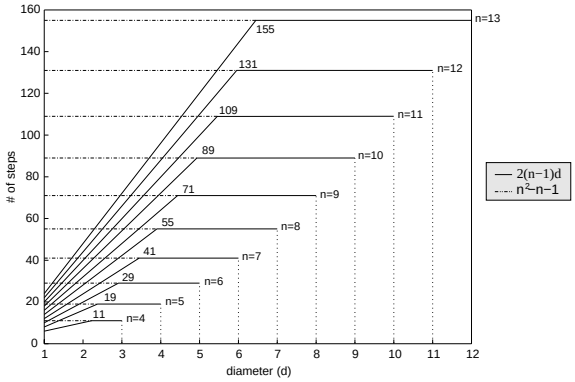

Figure 5. The bound as a function of the number of nodes and of the longest path in the network.

Theorem 2 *Let π be a network with a fixed topology matrix M. If the communication between the nodes is bidirectional, then the number of steps required to complete a consensus process is $\sigma(\pi, M) \leq 2(n-1)d(\pi, M)$.*

Proof. Let p_k be the node that triggers the consensus process and let p_w be the last node that receives that information from p_k. Under this assumption, A^w is the last vector to be updated to a non-null value. It takes no more than $(n-1)d$ for p_w to transmit its A^w vector after receiving a vector with the k-th flag marked. This is true because the worst case is when p_k and p_w are at the extremal sides of the longest path in the network, for which holds $\Delta_{hop}(p_k, p_w) = d$. Moreover, the maximum amount of steps needed to have a generic node transmitting after the transmission of a node directly linked to it is $n-1$ from Lemma 3. Note that if p_k is not placed at the extremal side of the longest path, because it is in the middle of such a path or even at the extremal side of a shorter path, then $\Delta_{hop}(p_k, p_w) \leq d$. When p_w receives a vector with the k-th flag marked, it updates its vector and later transmits it, in the right synchronization cycle. After that cycle, no more than $(n-1)d$ steps are required to propagate its information to all the other nodes. In particular, let p_z be the last node that completes its own vector, then $\Delta_{hop}(p_w, p_z) \leq d$ for the same reasons as above. Summing the contributions of the two-way broadcasts, i.e., $(n-1)d + (n-1)d$, yields the following bound $\sigma(\pi, M) \leq 2(n-1)d(\pi, M)$.

Since p_w is the last node receiving the flags information from p_k, when p_w starts to broadcast its updated vector all the other nodes have already received those flags from p_k and they have already started to broadcast their updated vectors, too. This assures that the flags broadcast by a generic node $p_i \in \pi$ are received by all the other nodes in the network before the flags from p_w are received by p_z. This results from the assumption that p_w is placed at the extremal side of the longest path, yielding $\forall p_i (p_i \in \pi \wedge \Delta_{hop}(p_i, p_z) \leq \Delta_{hop}(p_w, p_z))$. □

Definition 8 *The worst-case network topology for a given number of nodes n is the one in which a consensus process takes the highest number of steps to complete.*

Theorem 3 *If the communication among the nodes is bidirectional then the worst-case network topology is the one in which there is a single path $p_k \equiv p_{m_1} \leftrightarrow \ldots \leftrightarrow p_{m_{s-1}} \leftrightarrow p_{m_s} \equiv p_w$ where $s = n, \forall m_i, m_j (1 \leq m_i, m_j \leq n \wedge m_i \neq m_j)$, $m_s = m_{s-1} + 1$ and the consensus process is triggered by node $p_{m_s} \equiv p_w$. In this case, the number of steps required to complete a consensus process can be as high as $S(n) = n^2 - n - 1$.*

Proof. The topology depicted in Theorem 3 is a linear topology including all the nodes of the network. This is the worst-case topology because it implies the longest possible path with a given number of nodes ($d = n - 1$). Any other topology would imply the existence of forking nodes, i.e. nodes connected to more than two nodes. In such circumstances, the time to propagate any information from one extreme to the other can only be shorter. This is because, on one hand $d < n - 1$, necessarily, and on the other hand, after the forking node, the information flows in parallel over more than one link and thus, faster. If the node that starts the process is node $p_{m_s} \equiv p_w$, which lies at one extremal side of the path, to complete the process the information must first reach $p_k \equiv p_{m_1}$, which completes vector A^k, and then return back to p_w to allow it to also complete its vector A^w. This is the longest path that the information must cross. In this situation, from Lemma 2 we know that n steps are needed to cross a one-hop path in both directions, so $n(n-1)$ are needed to cross all the paths forward and backward from p_w to p_k. The last steps in the process, from m_{s-1} to m_s, can be avoided, since the process completes as soon as m_{s-1} transmits and m_s receives, i.e. no need to wait for m_s to transmit. The lowest number of steps that can be saved is 1, and it can only be achieved if $m_s = m_{s-1} + 1$. Summing all the contributions we have $n(n-1) - 1 = S(n)$. □

Notice that the bound given by Theorem 3 depends only on n and it establishes the absolute maximum number of steps that a consensus process may take with any topology and it is thus very practical. However, when $d \ll n$, that bound is also very pessimistic. Is such circumstances, the bound given by Theorem 2 is substantially tighter. Nevertheless, using this bound requires knowing d for the current topology, which can be determined inspecting the M matrix. Therefore, a better solution can be achieved by defining a new bound that corresponds to the lowest one, for each n, between the two ones previously referred. Such an improved bound is illustrated in Figure 5 where, for each n, the maximum number of steps is presented as a function of d. As an application example, consider the situation depicted in Figure 3. In that case, $n = 6$ and thus, applying Theorem 3, we know that any consensus process for 6 robots will terminate at most after $S(6) = 29$ synchronization steps. However, for that topology we know that $d = 2$.

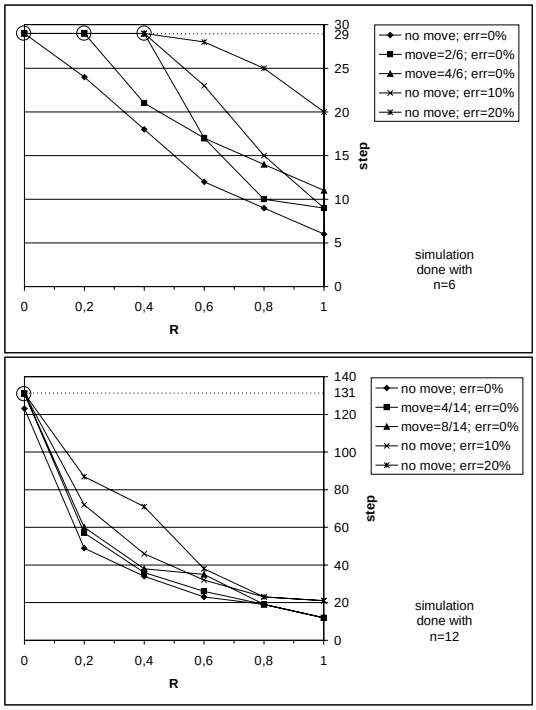

Figure 6. Simulation results with different combinations of mobility and errors.

Thus, applying Theorem 2 we deduce a tighter bound given by $2(n-1)d(\pi, M) = 20$ steps.

6 Simulation results

In order to assess the performance of the protocol, including when the nodes move and there are omissions of synchronization messages, we carried out several extensive simulations. The results concerning the number of steps actually taken to reach consensus are shown in Figure 6, using the maximum of at least 100.000 random topologies for each point. The topologies were generated considering two major cases, 6 nodes and 12 nodes, and always being fully connected. In order to classify the generated topologies we used R, the redundancy level of the network, as defined in Section 3. R gives an indication that is similar to the inverse of d, i.e., the larger R, the shorter the maximal distance in the network, and we used it for the sake of convenience in the generation of the topologies.

The lower curves show the number of steps in a favourable scenario, in the absence of errors and with a steady topology during the consensus process. In both major cases ($n = 6$ and $n = 12$), the number of steps actually reaches the upper bound for the case of $R = 0$, as expected, confirming the bound accuracy. As R increases, the number of required steps to reach a consensus rapidly decreases.

Then, we assessed the protocol under nodes mobility. The velocity of changes was roughly characterized by $move = X/Y$, meaning that X links were either broken or created in the topology matrix, every Y steps, during a consensus process. For $n = 6$, the results with $R = 0$ and 0.2 show that there were incomplete or partially complete processes (marked with a circle in the graph). For $n = 12$, such a situation happened for $R = 0$, only. For higher values of R, all processes reached consensus within the $S(n)$ upper bound.

Table 1 presents, in the last two columns, the actual percentage of processes that did not complete within the bound (partially-complete plus incomplete), and those that terminated incomplete, respectively, only for the cases in which those values were non-zero. The values show that such a percentage is already low for $R = 0$, becoming extremely low for $R = 0.2$, and zero for higher values. The column on "max n.c." shows the maximum number of vectors that did not reach consensus (this equals n when there were incomplete processes).

We also assessed the protocol behavior under omissions of the synchronization messages, according to the fault model described in Section 3. Therefore, for each n under test, we generated two cases: one case with 10% of random omissions with respect to the total number of synchronization messages in the process, and another case with 20% omissions. The results in terms of number of steps also show that for smaller R there are some incomplete or partially complete processes, as expected. Table 1 shows the actual numbers of partially complete (the "average p.c." column) and incomplete processes (the "average n.c." column).

The experiments show the robustness of the proposed protocol since, even in presence of relatively high mobility and errors, the consensus process completes within the $S(n)$ bound with a very high probability for $R > 0$. When it does not, the probability of terminating incomplete, which is the situation that generates greater disturbance, is very low, since most of such processes actually complete, but partially, only. This is expected because of the flooding nature of the protocol that makes use of all parallel paths in the topology. Thus, as long as there are some redundant paths, the resilience of the protocol increases substantially.

Finally, the results also show that increasing the number of nodes in the network increases its resilience to errors and mobility. This can be explained by the fact that for higher number of nodes the unfavourable topologies corresponding to $R = 0$ become less and less probable. Also, for the same R, there will be more redundant links if n is larger.

7 Conclusions

In this paper we proposed a new MAC level protocol to schedule real-time communications in a network of robotic mobile units over a wireless medium. It is based on the implicit EDF scheduling algorithm, which is collision-free, thus allowing high utilization of the medium bandwidth. The protocol addresses the problem of having a team of fully-connected, but not fully-linked network units and tolerates the presence of hidden nodes, either caused by exces-

n	R	change	err (%)	max n.c.	average p.c. (%)	average n.c. (%)
6	0	2/6	0	6	0.0739	0.0006
6	0	4/6	0	6	0.1169	0.0028
6	0.2	2/6	0	1	0.0001	0
6	0.2	4/6	0	1	0.0001	0
6	0	0	10	6	21.2620	18.2583
6	0	0	20	6	68.8619	35.6228
6	0.2	0	10	4	0.0769	0
6	0.2	0	20	6	1.5485	0.0003
6	0.4	0	10	1	0.0026	0
6	0.4	0	20	2	0.0034	0
12	0	4/14	0	12	0.0618	0.0020
12	0	8/14	0	12	0.0048	0.0040
12	0	0	10	8	0.3849	0
12	0	0	20	12	1.9560	0.0150

Table 1. Simulations results.

sive link lengths or by the presence of obstacles. The protocol uses global resource reservation to support dynamic changes in the global communication requirements under guaranteed timeliness. These changes may arise from external nodes that wish to join the team, from nodes that leave the team, either voluntarily or inadvertently (crash or movement), or from requests to change the current communication requirements.

The global resource reservation is based on a specific consensus process that uses periodic dissemination of state information. The main contributions of this work are the adaptation of implicit EDF for a dynamic environment and the design and analysis of the consensus process, including the determination of bounds for the maximum required number of steps to complete. The paper includes simulation results that show the effectiveness of the protocol even under transmission errors and nodes mobility.

The protocol is meant for small sets of mobile units, typically around 10. However, it can be integrated into a hierarchical scalable routing framework, using this protocol at the cell or zone level.

A positive characteristic of the proposed solution is that the period used for broadcasting system state information can be tuned to balance reactivity of the resource reservation mechanism and its bandwidth requirements. In fact, the longer the synchronization period, the longer the time required to agree on a decision, but the smaller the bandwidth required to transmit the system data. This is particularly relevant given the relatively large amount of system data that is exchanged via the synchronization message. Just as an example, in a scenario with 10 nodes and a CRT with 15 entries with 2 bytes parameter resolution, the total system data including clk, A and M would be around 150 bytes. Using a transmission rate of 1Mbit/s, T_{sync}=20ms, T_{slot}=770μs and a slot payload of 75 bytes would result in 26 slots per synchronization interval, 2 of which would be used for the system data representing a bandwitdh of 7.7%. In these circumstances, the reactivity of the resource reservation mechanism would be around 2s.

The framework within which this work developed includes current and future work to deal with the issues of clique formation, message routing, topology management and scalability. Particularly, there is a substantial attention dedicated to the use of the topology matrix to support routing of data messages, topology management controlling the movement of the robots to prevent $R = 0$ topologies, and management of channel reutilization to improve bandwidth efficiency.

References

[1] M. Caccamo, L. Y. Zhang, L. Sha, and G. Buttazzo. An implicit prioritized access protocol for wireless sensor networks. In *Proceedings of the IEEE Real-Time Systems Symposium*, Austin, Texas, December 2002.

[2] J.-D. Decotignie. Wireless fieldbusses – a survey of issues and solutions. In *Proc. 15th IFAC World Congress on Automatic Control (IFAC 2002)*, Barcelona, Spain, July 2002.

[3] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. 34(1):77–97, 1987.

[4] T. Facchinetti, G. Buttazzo, M. Caccamo, and L. Almeida. Wireless real-time communication protocol for cooperating mobile units. In *Proceedings of the 2nd International Workshop on Real-Time LANs in the Internet Age (RTLIA)*, Porto, Portugal, July 2003.

[5] M. Gerharz, C. de Waal, M. Frank, and P. Martini. Link stability in mobile wireless ad hoc networks. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN) 2002*, Tampa, Florida, November 2002.

[6] R. Grabowsky, L. E. Navarro-Serment, C. J. J. Paredis, and P. K. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, 8(3):293–308, June 2000.

[7] B. Hughes and V. Cahill. Achieving real-time guarantees in mobile ad hoc wireless networks. In *Proceedings of the Work-in-Progress session of the 24th IEEE Real-Time Systems Symposium*, pages 37–40, Cancun, Mexico, December 2003.

[8] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *j-TOPLAS*, 4(3):382–401, July 1982.

[9] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM*, 1(20):46–61, 1973.

[10] M. Pease, R. Shostak, and L. Lamport. Reaching agreements in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[11] T. Srinidhi, G. Sridhar, and V. Sridhar. Topology management in ad hoc mobile wireless networks. In *Proceedings of the Work-in-Progress session of the 24th IEEE Real-Time Systems Symposium*, pages 29–32, Cancun, Mexico, December 2003.

[12] J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou. Real-time communication and coordination in embedded sensor networks. In *Proceedings of the IEEE*, volume 91, pages 1002–1022, July 2003.

[13] J. Turek and D. Shasha. The many faces of consensus in distributed systems. *IEEE Computer*, pages 8–17, June 1992.

[14] J. Wu and I. Stojmenovic. Ad-hoc networks. *IEEE Computer*, 37(2), February 2004.