EURASIP Journal on Wireless Communications and Networking 2005:5, 712–730

Dynamic Resource Reservation and Connectivity Tracking to Support Real-Time Communication among Mobile Units

Tullio Facchinetti
Dipartimento di Informatica e Sistemistica (DIS), Università di Pavia, 27100 Pavia, Italy
Email: tullio.facchinetti@unipv.it

Giorgio Buttazzo
Dipartimento di Informatica e Sistemistica (DIS), Università di Pavia, 27100 Pavia, Italy
Email: buttazzo@unipv.it

Luis Almeida
Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA), and Departamento de Electrónica e Telecomunicações (DET), Universidade de Aveiro, 3810-193 Aveiro, Portugal
Email: lda@det.ua.pt

Received 29 June 2004; Revised 25 April 2005

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 in those applications in which the robots have to operate autonomously in unstructured environments. This paper proposes a MAC protocol for wireless communication that supports dynamic resource reservation and topology management for relatively small networks of cooperative units (10–20 units). 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. We also introduce a distributed connectivity tracking mechanism that is used to detect network partition and absent or crashed nodes. Finally, a set of simulation results are shown that illustrate the effectiveness of the proposed approaches.

Keywords and phrases: topology, wireless, mobile, real time, distributed network.

1. INTRODUCTION

The relevance of ah hoc networking is clearly stated by several authors (e.g., [1, 2]) that present specific applications suitable for mobile ad hoc networks (MANETs). One class of applications is the interconnection of multiple robotic mobile units. Groups of such units represent an attractive solution in those situations in which the environment's conditions are not suitable for direct human intervention. This can occur in space missions, in the exploration of hazardous environment, in demining, surveillance, and civil protection [3]. In these cases, relatively 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 [4].

Moreover, robots must exchange information concerning both the environment and their own state, which is

inherently time constrained. This calls for a real-time communication protocol capable of meeting the global communication requirements, namely, in terms of bandwidth and communication delays. However, achieving real-time communication over wireless networks has long been a challenge [5, 6] mainly due to the higher attenuation and higher bit error rates typical of that medium as well as its open character. 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 [5] 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 [6] because these are frequently structured, that is, based on fixed access points.

A further challenge in MANETs is supporting dynamic resource reservation as required by nodes that join or leave the team 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 and periodically broadcast state or environment information (e.g., a value of temperature, the concentration of a polutant, the position of a target, a video/audio stream, the robot's position, its energy level and integrity status). The underlying cooperation model follows the producer/consumer paradigm in which several producers transmit periodically information that is made available to consumers who may retrieve it from the network if required. This model is particularly adapted to applications such as teams of surveillance robots, rescue robots, or even soccer robots as those used in the RoboCup Middle Size League.

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 changes in resource allocation, enforcing globally coherent decisions. Moreover, to maintain updated information on the network topology even when nodes move, a similar mechanism based on a connectivity matrix is used to track the current topology. Both mechanisms, for consensus as well as for connectivity tracking, are the focus of this work.

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 introduces our approach to track the network topology. Section 5 describes the consensus procedure while Section 6 presents and validates an upper bound on the time taken by the consensus procedure and includes simulation results that show the effectiveness of the protocol even with errors and mobility. Section 7 illustrates the simulation results concerning the resource reservation method and the proposed topology-tracking algorithm. Some implementation issues are presented in Section 8, including an evaluation of the protocol overhead. Finally, Section 9 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, that is, based on fixed access points; and ad hoc. A further classification divides the latter category into mobile ad hoc networks (MANETs) [4] and sensor networks [5].

All categories have been extensively addressed by the research community but only a relatively small subset of the vast amount of the available literature addresses aspects related to real-time communication. Two fundamental aspects that constrain the real-time behavior are the medium access control (MAC) protocol and the mechanisms to handle dynamic communication requirements. This paper deals with these two aspects in the scope of MANETs, particularly for small teams of autonomous mobile robots, that is, with around 10 to 20 units, which move in the vicinity of each other and broadcast periodic information.

One of the main challenges in MANETs is dealing with mobility. In fact, as nodes move, the links between nodes may break and new links may be established, leading to a dynamic connectivity. To deal with mobility, MANETs typically use specific techniques. For example, in [8], 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 [9], 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. Combining 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. Soft real-time communication among a dynamic set of nodes, on top of IEEE 802.11 networks, is achieved in [10] by means of a dynamic bandwidth manager that adapts on line the transmission rates of current streams to accommodate new ones. However, 1-hop communication is considered, that is, a fully linked network, and the bandwidth manager is centralized in one node, collecting global information from the streams being transmitted. Conversely, [11] presents a scheme based on a modification of the IEEE 802.11 MAC, namely, distributed weighted fair

scheduling in which several streams are scheduled according to their weights by adequately adapting the backoff interval at the MAC level. The possibility for dynamic weights is also analyzed, allowing the use of such protocol in dynamic environments. Nevertheless, in these two solutions, the real-time properties of the protocols are relatively poor, with collisions still occurring, thus their soft real-time nature. Johansson et al. [12] address Bluetooth and, particularly, the impact of using several traffic scheduling policies by the piconet master to deliver real-time communication services. This protocol uses global information at the piconet level, which is kept centrally by the master to poll the remaining nodes for their transmissions.

This paper proposes the use of implicit EDF [13] to provide real-time guarantees to the network traffic while using nearly all the communication medium bandwidth. The price to pay is an extra overhead required for system synchronization. Implicit EDF is a time-triggered medium access control discipline in which all nodes implement in parallel an EDF queue of all communication requests. Collisions are avoided by replicating and executing the EDF scheduler in parallel in all nodes, in a tightly synchronized way. This means that all local EDF schedulers generate precisely the same schedule which corresponds to implementing a single global EDF queue of ready messages. In this model, every node knows when to transmit and receive, even in the presence of hidden nodes. The protocol uses a slotted framework in which messages are allocated an integer number of fixed duration slots.

Implicit EDF is further combined with a consensus procedure to support dynamic communication requirements and, generally, dynamic resource reservation. This is necessary to enforce simultaneous updating of all local EDF schedules. Moreover, a connectivity tracking mechanism is used that supports the detection of absent or crashed nodes.

The problem of reaching a consensus has been widely considered in the literature on distributed systems since it was firstly introduced in [14]. Dolev et al. [15] proved that in a system with clock synchronization and time-bounded communications, such as ours, it is possible to reach a consensus. An equivalent problem is the one of fault-tolerant broadcasts [16]. Many of the previously proposed algorithms [17, 18] are in principle applicable to a wireless distributed system, which can be seen as one using an unreliable communication medium. Consensus is thus achieved by exchanging specific messages, the number of which depends on the type and number of faults that are to be tolerated. In a wireless medium the number of faults can be substantial, for example, caused by transmission errors, interferences, and dynamic network topology. This makes achieving consensus in a wireless network typically bandwidth expensive.

Therefore, this paper proposes a consensus procedure that keeps the respective overhead under deterministic bounds and isolates it from the remaining traffic to prevent mutual temporal interference. This is achieved piggybacking the consensus-related information on top of a periodic system message used for synchronization purposes whose bandwidth is guaranteed.

The consensus procedure is optimistic in the sense that, upon a change request, a future time instant is defined at which the procedure is 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.

A preliminary combination of implicit EDF and the proposed consensus procedure was first presented in [19] but with the restrictive assumption of absence of hidden nodes, a restriction that is now lifted.

3. SYSTEM MODEL

System architecture

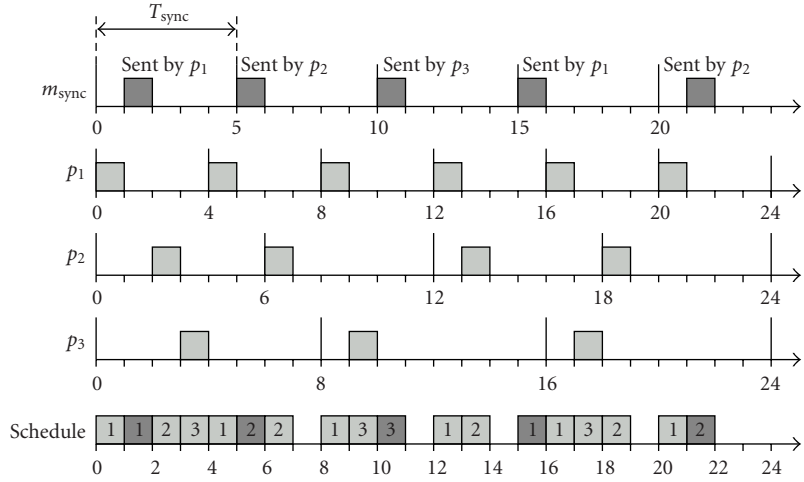
The global system architecture considered in this paper consists of a set Π of n_π mobile units or nodes, $\Pi = \{p_1, \ldots, p_{n_\pi}\}$, which can communicate over a radio-based wireless medium. Every unit is unambiguously identified by a statically assigned identifier $\mathrm{Id}(p_i) = \mathrm{Id}_i$. 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 node p_i is linked to node p_j if p_i is able to listen to a transmission from p_j. In such a case, we say there is a link L_{ij} from node p_i to node p_j, represented by the edge $p_i \rightarrow p_j$ in the connectivity 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 $p_i \equiv p_{m_1} \rightarrow \cdots \rightarrow p_{m_{s-1}} \rightarrow p_{m_s} \equiv p_j$. 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 at least a link between them.

In order to maintain topological information of the network at each instant, each node p_k uses a connectivity 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 ith row and jth 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 connectivity matrix owned by node p_k at instant t.

Communication model

Communication among nodes is organized in consecutive slots, referred to as system ticks, which have a constant duration T_{tick}. The model is periodic, which means that all message streams served by the communication system are



i	T	C
Sync	5	1
1	4	1
2	6	1
3	8	1

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

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 otherwise stated.

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 to 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 *ticks*. The communication requirements table (CRT) holds the properties of all the messages to be scheduled by the communication system, so $\text{CRT} = \{m_l(I_l, C_l, T_l, D_l, O_l),\ l = 1, \ldots, N\}$, where N is the number of message streams produced by all nodes. The total bandwidth requirement is given by $U_{\text{CRT}} = \sum_{l=1}^{N} C_l/T_l$.

We say that the traffic model is dynamic since existing network nodes may request changes in their message streams, or nodes not in the network 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 $\text{CRT}^k(t)$ 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 $\text{CRT}^k(t)$, $M^k(t)$, local clock value $\text{clk}_k(t)$, and other information related to the consensus procedure triggered upon CRT change requests. This is called the system synchronization message m_{sync} and it is broadcast by all nodes in a round-robin fashion $(p_k, \ldots, p_1, p_n, p_{n-1}, \ldots, p_{k+1})$. We will call the transmission of a synchronization message a *step*. 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 a 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, that is, $C_{1,\ldots,3} = C_{sync} = 1$, and the step period is 5, that is, $T_{sync} = 5$.

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

$$U_{tot} = \sum_{i=1}^{N} \frac{C_i}{T_i} + \frac{C_{sync}}{T_{sync}}. \tag{1}$$

Notice that U_{tot} includes all overheads, such as all the control information sent each slot, as well as any unused space within the slots.

Finally, the clock sent within the synchronization message $(\text{clk}_i(t))$ includes both a representation of continuous time (i.e., with microseconds resolution) and an absolute tick 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 $\text{clk}_i(t)$ to refer to the tick counter only, unless explicitly stated otherwise.

Real-time guarantees

As referred before, messages are scheduled using the implicit EDF approach [13]. 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, that is, between packets. Since all messages also become ready for transmission synchronously with the slot boundary, then, this scheduling model is equivalent to preemptive EDF [20]. Therefore, the following condition is sufficient and necessary to guarantee that the traffic is schedulable, that is, that all messages will be transmitted once within their periods:

$$U_{\text{tot}} \leq 1. \tag{2}$$

This condition assumes deadlines equal to periods and has the advantage of being extremely simple to evaluate. Other conditions exist, however, for the general case of arbitrary deadlines [21], that can be directly applied.

The above condition is evaluated on line, as part of an admission control, prior to accepting any change in the current communication requirements, for example, updating a period or adding a new stream. Changes are accepted if the condition is met, thus assuring a continued real-time behavior.

During topology changes the timeliness of transmissions is assured by means of the synchronization mechanisms of the EDF schedulers. However, the set of nodes that receive a given message might change. If a node needs a given stream that is no longer receiving, it must issue a request for the addition of one or more streams to relay the information of the former one. If n streams are added with period T, the end-to-end delay is upper bounded by $(n+1) * T - 1$. Tighter estimations can be achieved with a judicious use of offsets.

4. CONNECTIVITY TRACKING

This section presents the network connection tracking mechanism. Generally, due to mobility, crashes, or other phenomena, the connectivity matrices of different nodes will differ as soon as a change in the network topology occurs, since they do not all perceive that change directly. The proposed algorithm is based on the exchange of the connectivity matrix held by each node, supporting a convergence of all the matrices to the unique and correct view of the whole network links. The algorithm makes the simple assumption that all nodes are able to detect omissions of expected transmissions according to the current schedule. This assumption is easy to achieve in the proposed communication model, but does not limit the usage of our approach to such a communication model.

To spread the knowledge on the connections through the network and to achieve the covergence of the matrices owned by all the nodes to the right view of the network connectivity, each node p_w must broadcast its own connectivity matrix $M^w(t)$. When node p_k receives a broadcast or does not receive an expected transmission, it locally updates its own $M^k(t)$ matrix and a local state variable $\delta^k(t)$ according to Algorithm 1.

```
update matrix (k, w, M^w, δ^k)
(1)   if (p_k receives the expected M^w) {
(2)     d = φ(w, M^w)
(3)     for each i ≠ k {
(4)       if (d[i] + 1 ≤ δ^k[i] · dist) {
(5)         set column M_i^k = M_i^w
(6)         set δ^k[i] = (w, d[i]+1)
(7)       }
(8)       else {
(9)         if (δ^k[i] · node = w) {
(10)          set δ^k[i] = (NULL, ∞)
(11)        }
(12)      }
(13)    }
(14)    set M_wk^k = 1
(15)  }
(16)  else {
(17)    if (M_wk^k = 1) {
(18)      set δ^k[w] = (NULL, ∞)
(19)      for each i such that δ^k[i] · node = w {
(20)        set δ^k[i] = (NULL, ∞)
(21)      }
(22)    }
(23)    set M_wk^k = 0
(24)  }
(25)  for each i such that δ^k[i] · dist = ∞
(26)    set column M_i^k = 0
```

Algorithm 1: The updating algorithm for the connectivity matrix.

4.1. Data structures

Two data structures are used by each node p_k to track the exact topology of the team:

(i) the connectivity matrix $M^k(t)$ as described in the previous section;
(ii) the minimum distance vector $\delta^k(t)$.

The $\delta^k(t)$ is a vector of n elements where the ith vector element, that is, $\delta^k[i]$, contains the identifier of node p_w from which node p_k got the information about the links of node p_i; it also contains the distance (in terms of hops) of node p_w from node p_k. We will indicate the content of the $\delta^k[i]$ as $\delta[i]^k \cdot$ node for the node identifier and $\delta^k[i] \cdot$ dist for the value of the distance. We will also write $\delta^k[i] = (n, d)$ if $\delta^k[i] \cdot$ node $= n$ and $\delta[i]^k \cdot$ dist $= d$.

While the matrix M^k must be broadcast, the δ^k vector is stored and used locally to the nodes only. This is very convenient as M^k is a binary matrix and can be encoded in just a small number of bytes.

4.2. Updating algorithm

The following terminology is used to describe the algorithm:

(i) p_k is the node that updates its matrix M^k;
(ii) p_w is the node that broadcast its matrix M^w;
(iii) δ^k is the minimum distance vector owned by node p_k;
(iv) the function $\phi(w, M^w)$ returns the minimum distances of all the nodes reachable from node w, as a result of the inspection of matrix M^w.

Note that a matrix broadcasting may not be heard by node p_k depending on several factors: high distance, presence of obstacle between the nodes, limited transmission power, interferences, and so forth.

The algorithm for updating the connectivity matrix is illustrated in Algorithm 1.

The basic idea behind the algorithm is that when node p_k receives a matrix M^w, it extracts the information about the distances of the broadcasting node p_w to all the other nodes. Then, p_k updates the ith column of its own matrix M^k (that refers to the ingoing links of node p_i) only if p_w is closer to p_i than the previous node from which the information was taken. The distance of the previous node is retrieved by inspecting the $\delta^k[i] \cdot \text{dist}$ value. When p_k does not receive an expected broadcast from p_w, it resets all the columns in M^k that were taken from a previous reception of M^w (if any) and resets the entries stored in δ^k that refer to p_w as well.

4.3. Description of the algorithm

Firstly, we assume that $\delta^k[k] = (k, 0)$, meaning that node p_k is 0 hops distant from itself. We also make the nonrestrictive assumption that $M^k_{ii} = 0$ for all i.

We must distinguish between two situations: line (1) tests if an expected communication was received. If matrix M^w was received, then its content can be used to update M^k, else the local variables have to be updated in a different manner. From line (1) to line (16) we consider the case of matrix reception.

Line (2) calls the function $\phi(w, M^w)$ in order to analyze the received matrix and to calculate the minimum distances from p_w to all the nodes connected to it. It returns the vector d containing the minimum distances of node p_w from all the other nodes on the basis of the paths detected by inspecting M^w. By writing $d[i] = x$ we mean that node p_i is x hops far from p_w. In Section 4.4 we report a more detailed description of this function.

Line (3) starts the cycle for updating every column of p_k excluding the kth one, in which each flag is updated only on the basis of the matrix reception. Line (4) tests if, for each node p_i, node p_w is closer to p_i than the node from which the current data in the ith column was copied. If it is closer, the ith column is copied from M^w to M^k (line (5)) and the identifier of p_w, together with its distance from p_i, is stored in $\delta^k[i]$ (line (6)). In line (6) we add 1 to the value of $d[i]$ to take into account the distance between p_k and p_w (1 hop).

If the distance between p_w and p_i is greater than the one stored in the $\delta^k[i] \cdot \text{dist}$ and the sending node is equal to $\delta^k[i] \cdot \text{node}$ (line (9)), then we reset the $\delta^k[i]$ entries (line (10)). This is done in order to reset the knowledge of p_k in this particular case and to accept an update from a node closer to p_k; this is fundamental for the convergence of the algorithm when the node mobility causes the formation of separated subnetworks.

In line (14) the flag M^k_{wk} is set to keep track of the correct reception by p_k of the matrix sent by p_w.

From line (16) the algorithm deals with a missing reception of an expected matrix. The instruction at line (17) checks if the node that missed the transmission was registered as a 1-hop distant node (flag $M^k_{wk} = 1$). If so, the algorithm first resets the $\delta^k[w]$ entry (line (18)) together with all the entries of δ^k directly related to p_w (line (20)). Finally, it stores the information about the missed reception by unmarking the cell M^k_{wk} (line (23)).

Since during the execution of the algorithm so far some entries may have been set to (NULL, ∞), we have to clear the related rows. In line (26) we reset the ith column of M^k if $\delta^k[i] = \infty$.

4.4. Evaluation of the minimum distance

The function $d = \phi(i, M)$ is used to inspect the connectivity matrix M in order to get the minimum distances among node p_i and all the other nodes of the network on the basis of the paths defined by M. It returns a vector d where $d[j]$ represents the distance between p_i and p_j in number of hops. The distance from p_w to itself is 0. If there are any paths connecting p_i with another node p_j, then $d[j] = \infty$.

For the evaluation of the distances, $\phi(i, M)$ uses the breadth-first search (BFS) [22]. The function $\phi(i, M)$ is the most expensive computation performed by the connectivity tracking algorithm. While all the loops used to update the matrix have a complexity that is $O(n)$, if L is the total number of links among the nodes—bidirectional links are counted twice—the complexity of $\phi(i, M)$ is $O(nL)$.

4.5. Properties and usefulness of the matrix

The main benefit associated to the connectivity matrix is the simple determination of which nodes receive the transmissions of any other nodes. However, this requires a careful inspection of the matrix, mainly due to the possible existence of asymmetric links. The rows of a generic matrix M^w give information on the nodes that are received by node p_w; on the other hand, the columns of the matrix give information on the nodes that listen to a broadcast of node p_w. This property is evident in the examples reported (Figures 2, 3, and 4). Among other possible uses, this information can also be useful for routing to determine a good path (e.g., the shortest one) from source to destination. This can be achieved using a simple BFS search (Section 4.4).

In Figure 2 there is an example of a network connected with both unidirectional and bidirectional links. By examining the network topology, it is easy to check that from all the nodes there exists a path connecting all the other nodes in the two ways (ingoing and outgoing). This corresponds to a connectivity matrix without empty rows or columns.

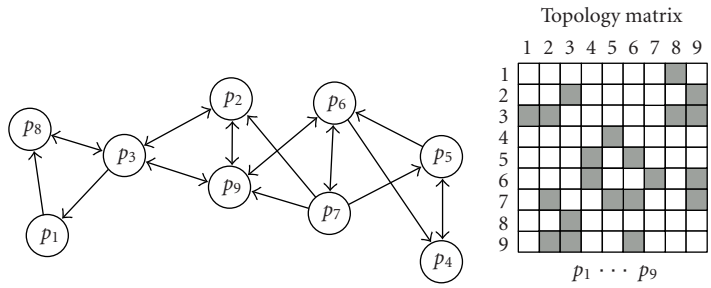


Figure 2: Example of unidirectional links between the nodes.

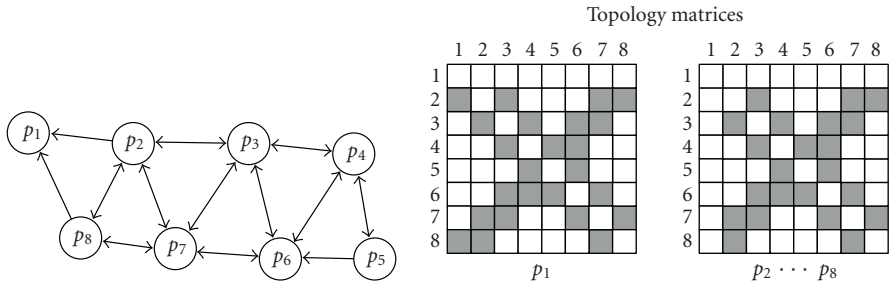


Figure 3: Example of isolated node: node p_1 broadcast does not reach any other nodes.

Another use of the connectivity matrix is to identify isolated nodes, for example, due to insufficient transmission range, or also network partitions. If node p_w cannot be heard by the other nodes in the network, then its matrix M^w presents an empty wth column. In the same way, all the nodes will present an empty wth row. The situation is well depicted in Figure 3. While most of the nodes in the network are connected with bidirectional links, node p_1, due to its position or transmission range, can only receive messages from the other nodes: the column 1 of M^1 is empty. The matrix of all the other nodes, M^i with $i = 2,\ldots,8$, have the row 1 empty, since they did not receive any transmission from node p_1.

Finally, another very interesting property of the proposed connectivity tracking algorithm is the speed of detecting absent nodes. The indentifier of the nodes that deliver the information is stored inside the MDV vector, as well as the distance from the node that is currently consuming such information. This implies a very useful property: a node p_i that was directly connected (through a 1-hop link) to a node p_j is able to detect the absence of node p_j, due to crash or insufficient transmission range, as soon as it detects the omission of the respective broadcast. This happens because $\mathrm{MDV}^k[i] = 1$, meaning that the distance value referring to p_j and stored into MDV is 1, which is the minimum possible value for any $j \neq i$. As a corollary of the previous property, a node can check if it is isolated from the remaining nodes in only one synchronization round (n steps), that is, when it detects the omission of the broadcasts from all the other nodes in the network.

5. 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 and locally to 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 broadcast 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 ith 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 acknowledgment of the global awareness of the decision to be taken at a defined time in the future.

5.1. The consensus process

In the field of distributed systems, there is a substantial amount of work in consensus processes, which must generally enforce the following three properties [17]: termination, validity, and agreement. Below, we state these properties in

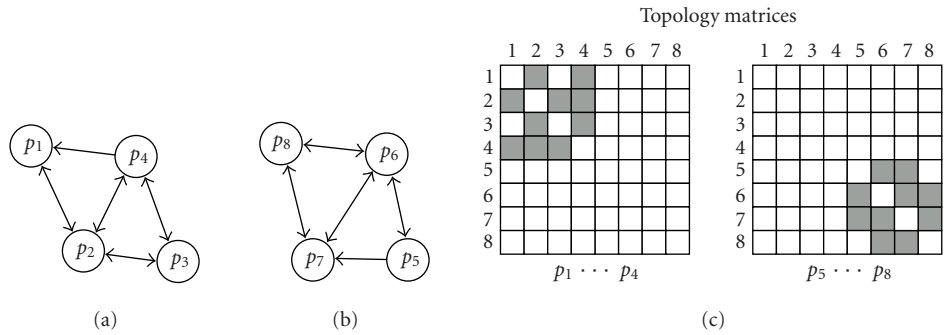


Figure 4: Example of matrix configuration for partitioned networks.

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.
(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 inconsistency does not jeopardize the consistent operation of the system. This is enforced by a positive aggregated acknowledgment of the consensus process in all nodes that allows differentiation of 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 acknowledgment is based on the agreement vector A.

5.2. Triggering a new process

When a node p_k needs to trigger a consensus process, it must fulfill the following.

(1) It must 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, that is, $\text{proc}_k = \text{clk}_k(t)$. Recall that $\text{clk}_k(t)$ is the tick counter value of the slot in which m_{sync} is sent.
(2) It must 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 this case, p_k cannot start a new process, which must be retriggered later.
(4) Otherwise, or after the termination of the previous process, it must mark the cell A_k^k in an empty (new) vector.
(5) It must 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 5 can receive requests from p_{new2} and p_{new3}).
(6) It must set the agreement time t_a equal to the triggering time $\text{clk}_k(t)$ plus an upper bound on the duration of the consensus process, as derived further on $(S(n)T_{\text{sync}})$. 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) It must send the synchronous message m_{sync} with the updated agreement information, that is, proc_k, Id_i, A, t_a, together with the communication requirements update, that is, the properties of the message to be adapted, added, or removed.

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, e.g., invalidate the update instant). 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 5, 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 reissued later.

5.3. Updating the agreement vector

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

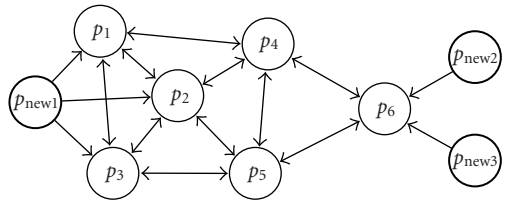


Figure 5: Example of simultaneous starts of multiple consensus processes.

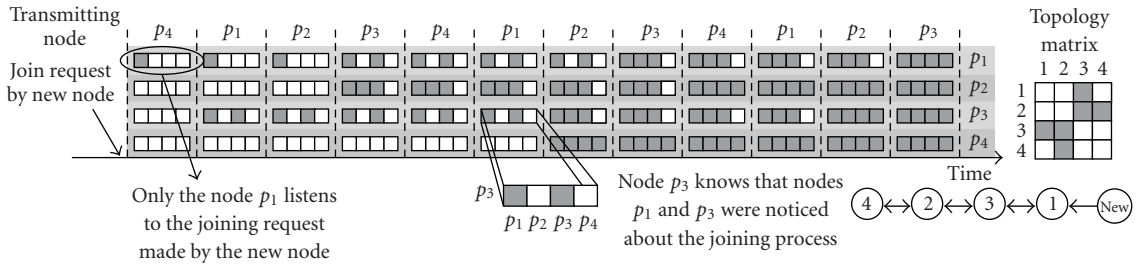


Figure 6: Example of the agreement vector update.

(1) If node p_k is not currently engaged in any consensus process, that is, A^k is empty, it performs the following operations:
   (a) $A^k_k = 1$,
   (b) $A^k = A^k \mid A^w$.
(2) Otherwise, node p_k is currently engaged in one on-going agreement process, that is, A^k is not empty, then it must check whether the received vector corresponds to the same process or a different one.
   (a) If $\text{proc}_k = \text{proc}_w$, then it is the same process and thus p_k updates its vector with the received one: $A^k = A^k | A^w$.
   (b) Else if $\text{proc}_k < \text{proc}_w$, 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) Else if $\text{proc}_k > \text{proc}_w$, 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, that is, $A^k_k = 1$.

The | operator in rules (1b) and (2a) means a *bitwise or* and 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 (2b) refer to situations in which p_k is notified of 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 reissue the request at a time after the agreement time of the on-going consensus process. An example of vector updates during an agreement process is depicted in Figure 6.

5.4. Termination of a consensus process

As mentioned in Section 5.2, 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. In the absence of errors, broken links and crashes or absent nodes, it is possible to prove (presented in Section 6) that at time t_a, whichever the current network topology is, the process will be *complete*.

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 for all $i, j = 1, \ldots, n$, $A^i_j = 1$.

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.

Firstly, 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 there exists i such that for all $j = 1, \ldots, n$, $A^i_j = 1$.

Notice that this is still a coherent situation, despite some nodes not knowing it. Therefore, those that reached the consensus, that is, have a fully marked A vector, execute the request relative to the consensus process. On the other hand,

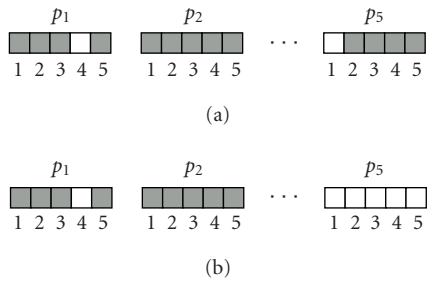


Figure 7: Example of errors in the vector broadcasting.

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 7a where node p_2 reach 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.

Figure 7b illustrates an impossible situation because if node p_5 holds an empty A vector, then the 5th 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*.

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 for all i, there 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 (see Section 8 on implementation issues) using the previous state of the CRT, that is, 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 connectivity matrix M referred in Section 3. In fact, a crashed or absent node is reflected in the connectivity 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.

5.5. Adding new nodes

The purpose of the consensus process is to support a global agreement on actions that have implications on global resources such as bandwidth. Namely, it was designed to support team formation, allowing new nodes to join, removal of nodes from the team, and changes in the global communication requirements. The latter two actions are triggered by nodes within the team. Therefore, they are already included in the m_{sync} round-robin circulation and they can submit their request when appropriate. On the other hand, the former action is triggered by the new node, that is, a node outside the team, which is not included in the current communication schedule. Thus, a special mechanism is required in this case, which is explained below.

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 the second is 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. Any team member that receives the request, when it comes to its time to transmit the m_{sync} message, initiates an agreement process as described in Section 5.2.

Following the 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 reissues the request. Possible duplicates of the request received by neighbor team nodes may generate parallel consensus processes, but only the oldest is kept, as discussed in Section 5.3.

6. 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 5. *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 nonempty 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 6. *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 5, 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. □

6.1. Upper bound on the number of steps

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 12 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 unfavorable topology, referred to as worst-case topology, and thus it holds equally for any possible linked topology. We firstly introduce the following definition.

Definition 7. 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

$$\Delta_s(p_k, p_w) = \begin{cases} w - k & \text{if } k \le w, \\ n + w - k & \text{if } k > w. \end{cases} \tag{3}$$

The step distance introduced in Definition 7 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.

Lemma 8. *For all $i, j (1 \le i, j \le n \wedge i \ne j)$, $\Delta_s(p_i, p_j) + \Delta_s(p_j, p_i) = n$.*

Proof. The proof follows directly from Definition 7. □

Lemma 9. *For all $k, w = 1, \ldots, n$, $\Delta_s(p_k, p_w) \le n - 1$.*

Proof. If $k = w$, then $\Delta_s(p_k, p_w) = 0 \le n$. If $k \ne w$, then $\Delta_s(p_w, p_k) \ge 1$ and $\Delta_s(p_k, p_w) \le n - 1$, from Lemma 8. □

Definition 10. Let $p_k \equiv p_{m_1} \to \cdots \to p_{m_{s-1}} \to p_{m_s} \equiv p_w$ be a path from p_k to p_w. The following distances are defined:

$$\begin{aligned} \Delta_{\text{hop}}(p_k, p_w) &= s - 1, \\ \Delta_t(p_k, p_w) &= \sum_{i=1}^{s-1} \Delta_s(p_{m_i}, p_{m_{i+1}}), \\ \Delta_r(p_k, p_w) &= \sum_{i=1}^{s-2} \Delta_s(p_{m_i}, p_{m_{i+1}}). \end{aligned} \tag{4}$$

The distance $\Delta_{\text{hop}}(p_k, p_w)$ 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. The distance $\Delta_r(p_k, p_w)$ specifies the number of steps required for p_w to receive an information that was initially sent by p_k. Note that such information is sent to $p_w \equiv p_{p_s}$ from node $p_{p_{s-1}}$.

Definition 11. Let π be a network with a connectivity matrix M. We say that $\mu_{\text{hop}}(\pi, M)$ is the maximal distance (or *diameter*) in the network between two nodes if and only if for all $i, j = 1, \ldots, n$, $\Delta_{\text{hop}}(p_i, p_j) \le \mu_{\text{hop}}(\pi, M)$.

Theorem 12. *Let π be a network with a fixed connectivity matrix M. If the communication between the nodes is bidirectional, then the number of steps required to complete a consensus process is $\sigma(\pi, M) \le 2(n-1)\mu_{\text{hop}}(\pi, M)$.*

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 $(n-1)\mu_{\text{hop}}$ for p_w to transmit its A^w vector after receiving a vector with the kth 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 $\Delta_{\text{hop}}(p_k, p_w) = \mu_{\text{hop}}$ from Definition 11 holds. 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 9. 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 $\Delta_{\text{hop}}(p_k, p_w) \le \mu_{\text{hop}}$. When p_w receives a vector with the kth flag marked, it updates its vector and later transmits it, in the right synchronization cycle. After that cycle, no more than $(n-1)\mu_{\text{hop}}$ 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 $\Delta_{\text{hop}}(p_w, p_z) \le \mu_{\text{hop}}$ for the same reasons as above. Summing the contributions of the two-way broadcasts, that is, $(n-1)\mu_{\text{hop}} + (n-1)\mu_{\text{hop}}$, yields the following bound $\sigma(\pi, M) \le 2(n-1)\mu_{\text{hop}}(\pi, M)$.

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

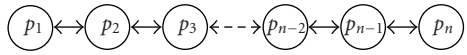


Figure 8: Worst-case network topology.

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, for all $p_i \in \pi, \Delta_{\text{hop}}(p_i, p_z) \ldots$. □

Definition 13. 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 14. *If the communication among the nodes is bidirectional, then the worst-case network topology is the one in which there is a single path* $p_k \equiv p_{m_1} \leftrightarrow \cdots \leftrightarrow p_{m_{s-1}} \leftrightarrow p_{m_s} \equiv p_w$ *where* $s = n$, *for all* $m_i, m_j (1 \leq m_i, m_j \leq n \wedge m_i \neq m_j)$, $m_s = m_{s-1} + 1$, *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*

$$S(n) = n^2 - n - 1. \tag{5}$$

Proof. The topology depicted in Theorem 14 is a linear topology including all the nodes of the network (Figure 8). This is the worst-case topology because it implies the longest possible path with a given number of nodes ($\mu_{\text{hop}} = n - 1$). Any other topology would imply the existence of forking nodes, that is, 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 the one hand $\mu_{\text{hop}} < n - 1$, 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 8 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, that is, 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 14 depends only on n and it establishes the absolute maximum number of steps that a consensus process may take with any topology

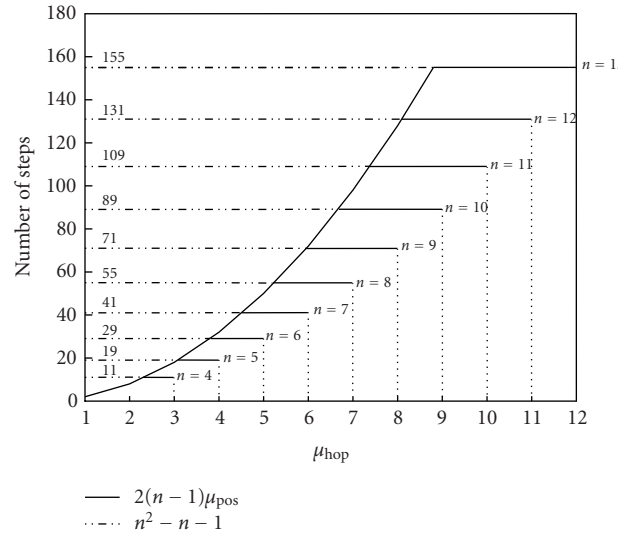


Figure 9: The bound as a function of the number of nodes and of the longest path in the network.

and it is thus very practical. However, when $\mu_{\text{hop}} \ll n$, that bound is also very pessimistic. Is such circumstances, the bound given by Theorem 12 is substantially tighter. Nevertheless, using this bound requires knowing μ_{hop} 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 to. Such an improved bound is illustrated in Figure 9 where, for each n, the maximum number of steps is presented as a function of μ_{hop}.

As an application example, consider the situation depicted in Figure 5. In that case, $n = 6$ and thus, applying Theorem 14, 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 $\mu_{\text{hop}} = 2$. Thus, applying Theorem 12, we deduce a tighter bound given by $2(n-1)\mu_{\text{hop}}(\pi, M) = 20$ steps.

7. SIMULATION RESULTS

This section shows separate results for the agreement process and the topology management approach.

7.1. Agreement process

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 10, 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

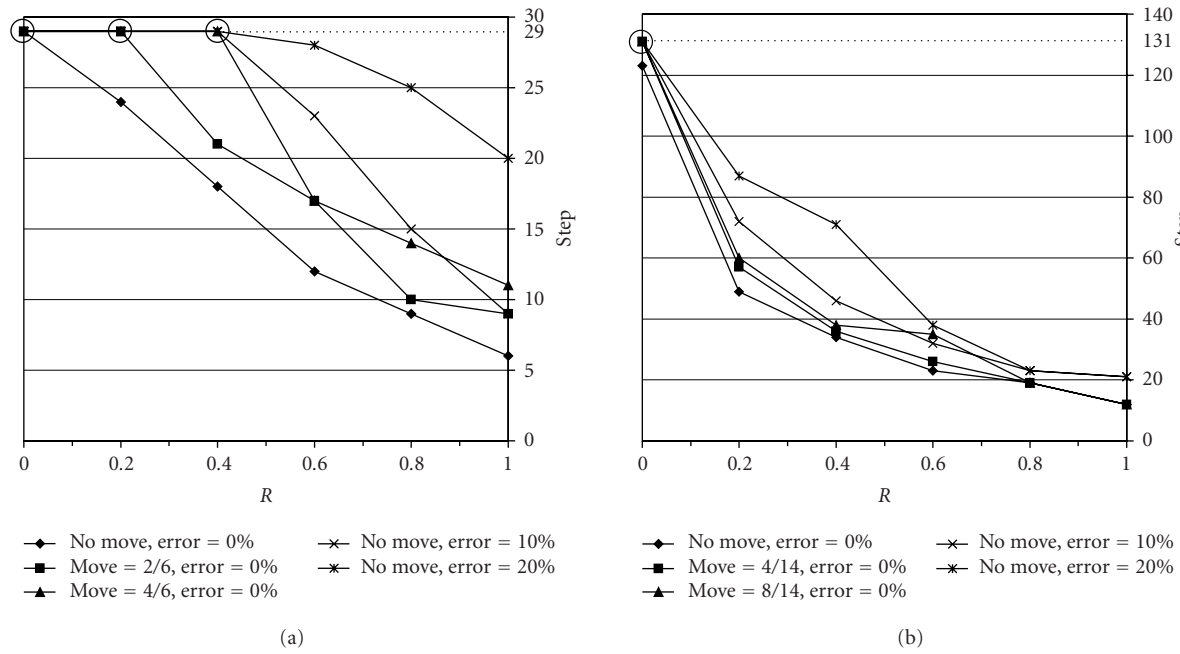


Figure 10: Simulation results with different combinations of mobility and errors: (a) $n = 6$, (b) $n = 12$.

connected. In order to classify the generated topologies we used R, the redundancy level of the network. This is defined as the ratio between the actual number of links of a given topology over the maximum number of links for the same set of nodes. Both terms of the ratio only count the links beyond the minimum required 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. R is similar to the inverse of μ_{hop}, that is, the larger R is, the shorter the maximal distance in the network is, 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 favorable scenario, with absence of errors and 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 $move = X/Y$. This means that X links were either broken or created in the connectivity matrix every Y steps during a consensus process. For $n = 6$, the results with $R = 0$ and 0.2 show that there were incompleted or partially completed processes (marked with a circle in the graph). For $n = 12$, such 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 completed plus incompleted), and those that terminated incompleted, respectively, only for the cases in which those values were nonzero. The values show that such percentage is already low for $R = 0$, becoming extremely low for $R = 0.2$ and zero for higher values. The column on "max no. of vectors not completed" shows for every process the maximum number of vectors that did not reach consensus (this equals n when there were incompleted processes).

We also assessed the protocol behavior under omissions of the synchronization messages, according to the fault model 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 incompleted or partially completed processes, as expected. Table 1 shows the actual numbers of not completed and incompleted processes.

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 incompleted, 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 unfavorable 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.

Table 1: Simulations results.

n	R	Changes	Errors (%)	Max steps	Max no. of vectors not completed	Average partially completed (%)	Average incompleted (%)
6	0	2/6	0	29*	6	0.0739	0.0006
6	0	4/6	0	29*	6	0.1169	0.0028
6	0.2	2/6	0	29*	1	0.0001	0
6	0.2	4/6	0	29*	1	0.0001	0
6	0	0	10	29*	6	21.2620	18.2583
6	0	0	20	29*	6	68.8619	35.6228
6	0.2	0	10	29*	4	0.0769	0
6	0.2	0	20	29*	6	1.5485	0.0003
6	0.4	0	10	29*	1	0.0026	0
6	0.4	0	20	29*	2	0.0034	0
12	0	4/14	0	131*	12	0.0618	0.0020
12	0	8/14	0	131*	12	0.0048	0.0040
12	0	0	10	131*	8	0.3849	0
12	0	0	20	131*	12	1.9560	0.0150

7.2. Connectivity tracking

In what concerns the topology management, we carried out a set of experiments to characterize the performance of the connectivity tracking system, trying to assess how fast the system can converge to the correct topology upon a change, and for how long, given a mobility model, the node connectivity matrix matches the correct one.

In a first set of experiments we addressed the speed of convergence to the correct topology, after a change in the network links. A set of nodes (n is equal to 6 and 12 in Figures 11 and 12, resp.) is randomly deployed in the environment. The position of the nodes is fixed, the network is fully connected and characterized by a given redundancy level R that varies from 0 to 1 in steps of 0.2 units.

In order to also test the speed of convergence in an initial state with an empty connectivity matrix, all nodes in this experiment start with their matrices cleared. Then, we firstly measure the number of steps needed to make all nodes matrices converge to the real network topology. These results are displayed in the graphs of Figures 11a and 12a. After that, a randomly chosen node is forced a crash, meaning that it stops transmitting until the end of the simulation. In this case we measure the number of steps needed to converge to the new topology. These measurements are displayed in Figures 11b and 12b. All the graphs are "box-and-whisker" plots for multiple redundancy level settings with a given number of nodes. They include the median values, which appear inside a rectangle that represents the number of measures between the first and third quartiles, as well as a pair of bars connecting the extreme values.

The first observation is that the number of steps needed to reach the right matrix is always lower than the worst-case bound obtained for an agreement process. This means that the topology management approach can be effectively used to monitor absent or crashed nodes during an agreement process, eliminating the problem of the communication disruption described in Section 5.4: if a node can not complete the agreement vector (i.e., due to crashes), it can consider the agreement as reached if it detects the absence of the nodes that did not participate in the process.

As we expected, the maximum and the median number of steps needed to converge decreases quadratically with the redundancy level of the network for the first part of the experiment (graphs on the left side). In the same graphs, the minimum values present an interesting anomaly: for redundancy levels of 40%, 60%, and 80%, they are lower than the value for the best connectivity scenario ($R = 100\%$). This can be easily understood realizing that, for $R = 100\%$, a fixed number of steps is needed to make all the nodes converge (exactly $2n - 1$ steps), and that the cells in the correct connectivity matrix are all marked. For lower but sufficiently high redundancy levels, the distance between the farthest nodes in the network is still low and many cells in the correct connectivity matrix are not marked. Since each node starts from an empty connectivity matrix, it can happen that all the right cells become marked in a number of steps lower than that required to mark all the cells.

Notice that, once the connectivity matrix converged, the number of steps needed to reconstruct the topology after a node crash is less, in the average, than that needed for building the connectivity information from scratch. This phenomenon is illustrated in Figures 11b and 12b, which show the number of steps required for convergence after a node crash. Notice that the speed of convergence after a crash depends on several factors, such as the position of the crashed node (which could cause the network to split into two or more subnetworks) and the interval between the time of crash and the next slot allocated to the crashed node.

7.3. Mobility test for the connectivity tracking method

While in the previous section we addressed the speed of convergence of our topology management system upon an

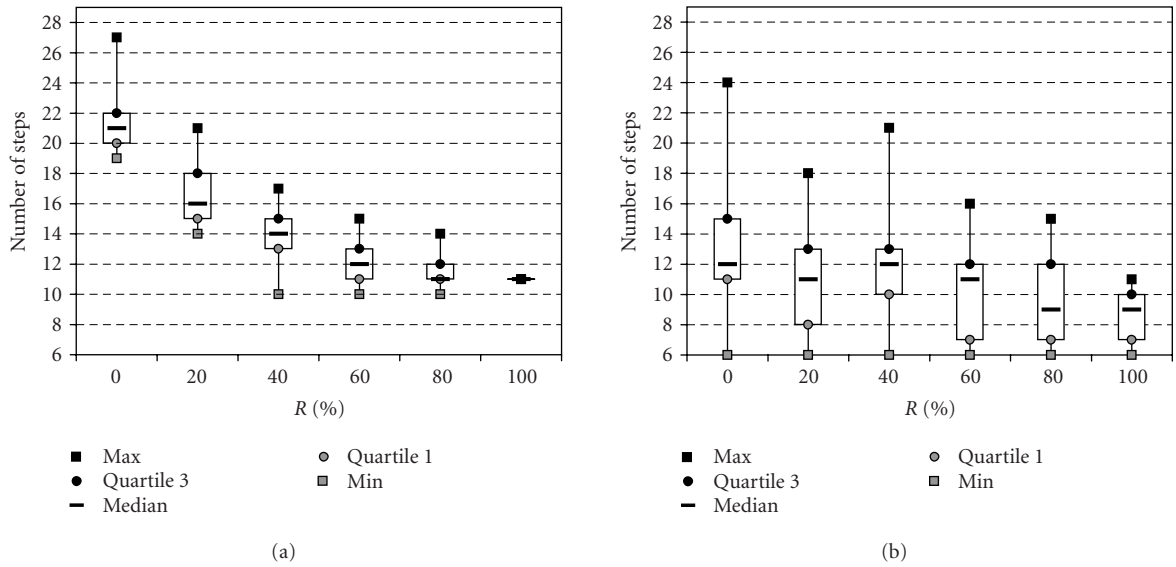


Figure 11: Box-and-whisker plots for the random crash test with multiple redundancy configuration settings with $n = 6$.

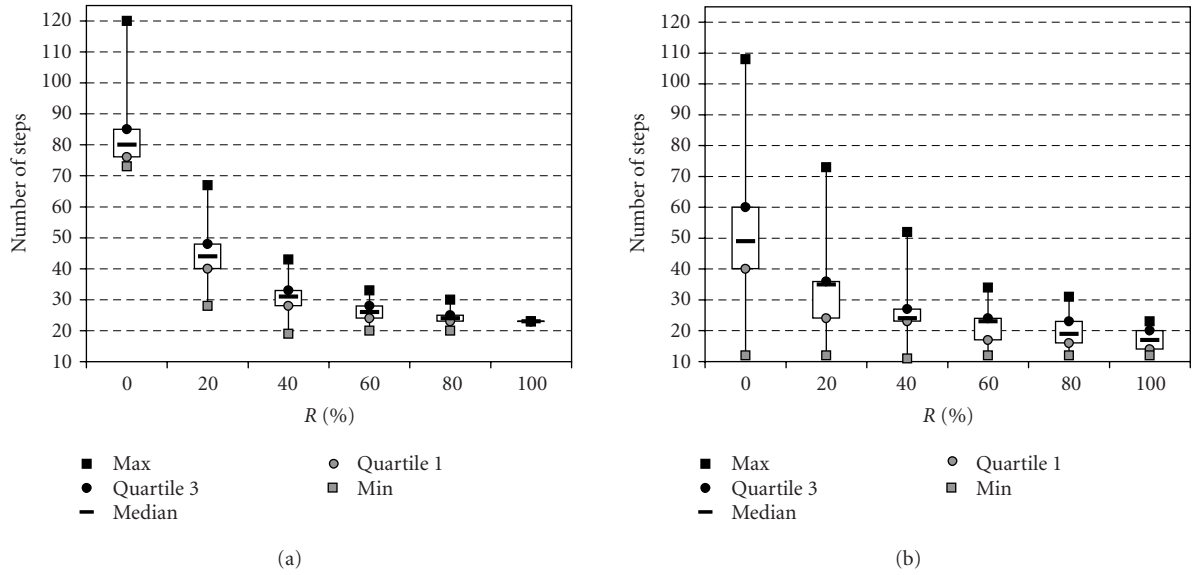


Figure 12: Box-and-whisker plots for the random crash test with multiple redundancy configuration settings with $n = 12$.

instantaneous change in the network, which remained stable after that change, in this section we address the behavior of the system with a dynamic topology as it is the case with moving nodes. We assumed that the nodes are moving as specified by the *Gauss-Markov* mobility model, which was firstly introduced in [23]. We also tested the protocol with other mobility models available in the literature, like *random way point* proposed in [24] and its variations in [25], but we show results for the Gauss-Markov model only, because it generates more adequate velocity patterns, with smooth variations in the speed and direction of the nodes. We assumed a variable number of nodes moving in a squared area of 50×50 m. Two values were considered for the speed of the nodes, 1 m/s and 2 m/s, which are reasonable values for robot's motion scenarios. For the radius of transmission ranges, we considered three different values, 10 m, 25 m, and 45 m. The standard deviation for the speed and the angle in the *Gauss-Markov* model were 0.1 m/s and $\pi/8$, respectively. Finally, the synchronization message is assumed to be broadcast every 50 milliseconds, which is a good compromise between bandwidth utilization and management information refreshing rate. Notice that decreasing such a value (down

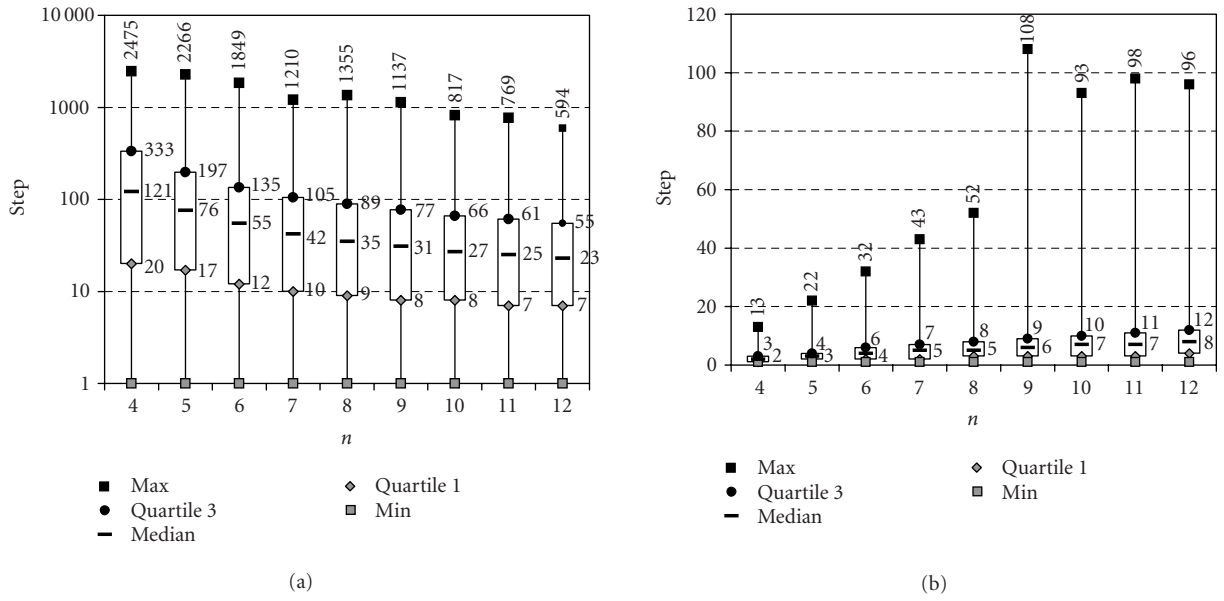


Figure 13: "Box-and-whisker" plots for the mobility test with speed of 1 m/s, transmission radius of 25 m, and different values of n.

to 10 milliseconds is practical) involves a faster updating frequency, thus leading to better results in all the tests that were performed.

To evaluate the performance of the connectivity tracking algorithm we let the system run for a time of 5–7 hours, which is rather long if compared with the timing characteristics involved. At every updating step we compared the connectivity matrices owned by all the nodes and the instantaneous real topology of the network. For every node, we obtained a temporal sequence (with granularity equal to the synchronization period) of similarity values. By convention, we consider a value of 0 meaning that a connectivity matrix represents exactly the real network topology, and 1 elsewhere. So, notice that even if only one link does not match the real topology, the matrix is considered to be wrong. Notice also that such a link may often be irrelevant for the packet routing or the support to the agreement process, especially with a high number of nodes in the network. At this point, we measured the length of the chains made by sequences of identical values. A sequence of 0 values indicates a time period of stability for the connectivity matrix, and the length measures the time that the connectivity matrix remains stable. A sequence of 1 values indicates a period of instability for the matrix.

The distribution of the lengths of the sampled sequences is reported in Figure 13. In particular, the two graphs show the distributions for the chains of matching sequences (Figure 13a) and nonmatching sequences (Figure 13b). We first describe the graph in Figure 13a. The fact that there exist stability sequences with very different length (the minimum length is 1) is an undesirable property of the distribution, because short times of stability involve little chance to have a right view of the network for a sufficiently long time (which is useful for routing). The values of the first quartile show that only 25% of the sequences are too short to be useful, while the remaining (75%) could be considered long enough to be used for the path prediction or to support the agreement process. We can also notice how all the values, in particular the median and the quartile values, decrease as the number of nodes is incremented. The reason is that, as the number of the nodes increases, both the probabilities for a link establishment and disruption during the node motion increase. It is worth observing that the negative effect of a nonmatching link has more influence when the number of nodes is low. In this sense, our statistics can be considered very pessimistic, leaving space to further improvements for networks with high number of nodes.

Figure 13b reports the statistics about the sequences of nonmatching matrices. It is relevant that, despite that the maximum length of a nonmatching sequence is in the order of n^2, the values of the third quartiles indicate that the 75% of the sequences have a length less than or equal to n.

In order to have an overall view of the topology during nodes motion, the results of both graphs can be merged and compared. Basically, the sequences of matching/nonmatching chains are typically made by very long matching chains ending in very short nonmatching chains. Within such a general behavior, short matching chains and few relatively long nonmatching chains can be found.

The global behavior of the algorithm and the proposed round-robin approach is summarized in Figure 14, which illustrates the percentage of time during which the connectivity matrices can be considered reliable for different speeds and communication ranges. Results with solid lines sum all the contributions from the matching chains (without considering the length), whereas those with dotted lines take into account the sum of the contributions of the chains with length greater than $2n$, for a given n. They are a more reliable

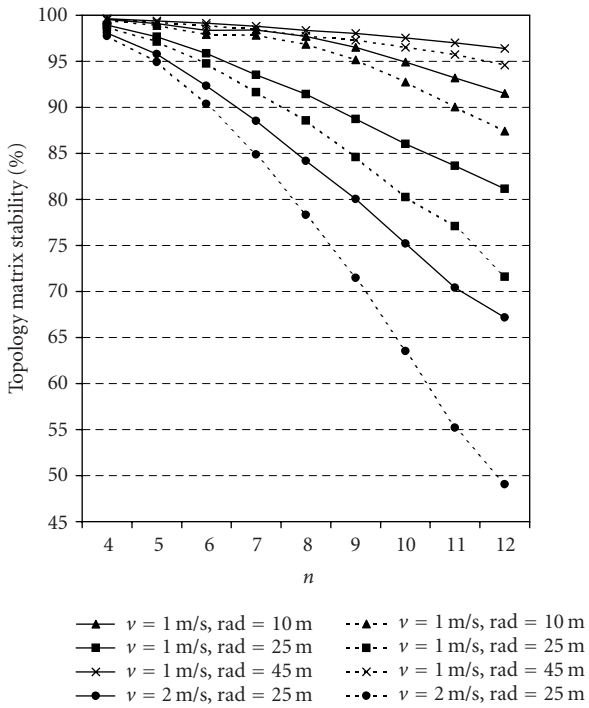


FIGURE 14: Percentage of time in which the connectivity matrix is stable during the mobility test as a function of n. Solid lines sum the contributions from matching chains, whereas dotted lines sum those chains with length greater than $2n$.

estimate of the quality of the topology approximation, and could be particularly useful for packet routing.

The curves highlight the higher influence of the node speed with respect to the transmission range. Such a behavior has been observed in several simulation experiments, not reported in this paper for lack of space.

Different transmission ranges were considered in the experiment: a short range of 10 m (20% of the 50×50 m moving area), a medium range of 25 m (50%) and a long range of 45 m (90%). The worst-case situation was observed for a medium range. This can be explained considering that for a short transmission range the nodes are isolated most of the time, whereas for a long range they are strongly connected most of the time. In fact, in these two extreme situations, the links are kept in the same state (broken or established) for long periods, thus the length of the stable chains is longer.

The strictly decreasing trend of the curves outlines the dependency of our approach on the number of nodes. This is, again, related to the time required to complete a full round of transmissions of the synchronization message, which is proportional to the number of nodes.

8. IMPLEMENTATION ISSUES

To deploy the protocol proposed in this paper there are several additional aspects to consider. One regards clock synchronization, which is fundamental to support the proper functioning of implicit EDF. The method to achieve clock synchronization, however, is independent of our protocol and several possibilities exist, for example, the fault-tolerant average algorithm used in TTP/C, the IEEE 1588 standard (master-slave), or even GPS whenever the operational environment allows its usage.

Another aspect concerns handling inconsistencies in the EDF schedulers that may arise for some unforeseen reason. To detect them, the CRTs are transmitted with a timestamp of the last update. Therefore, whenever a node receives, the received table is more up to date than its own. In that case, it replaces its CRT with the one just received and continues operation.

The startup procedure is another practical aspect of major importance. In the current stage, a special node, called the team leader, starts transmitting its synchronization message after detecting silence for longer than a given timeout. This allows other waiting nodes to join, one by one, building up the team. However, to prevent the single point of failure formed by the team leader, a fully distributed startup procedure with automatic election of team leader is currently being designed.

Finally, it is also necessary to set the protocol operational parameters. Two fundamental parameters are T_{tick} (the slot duration) and T_{sync} (the synchronization step). The former has a deep impact on the protocol data efficiency because, if it is much larger than the average length of the messages to be transmitted, a substantial part of the bandwidth is wasted. On the other hand, if it is too short, each message will always require the use of several slots, being broken into several slot packets and increasing the protocol overhead. Moreover, T_{tick} must include a guarding window between consecutive slots to account for possible clock drifts among nodes. The duration of this window should be equal to twice the precision achieved by the clock synchronization.

Consider a transmission rate of 1 Mbps, which is becoming typical with modern RF transceivers such as the model CC2400 of Chipcon, used in Bluetooth devices. With a clock precision of 50 microseconds, we can define guarding windows of 100 microseconds. If we define T_{tick}=1 milliseconds, we can transmit 900 bits per T_{tick}. Using 2-byte cyclic redundancy code (CRC), 2-byte message identifer, plus 2-byte preamble and control results in 48 bits of protocol control information, leaving 852 bits, or 106 bytes for data payload. This value seems a good compromise for a variety of applications and, if it is still too large, then nodes can aggregate data into one single message to make a more efficient use of the bandwidth.

In what concerns T_{sync}, it impacts directly on the reactivity of the system to global change requests. In fact, T_{sync} establishes the synchronization step, and thus the duration of the consensus processes is directly proportional to this value. Moreover, T_{sync} also determines the overhead introduced by the periodic transmission of m_{sync}.

If a value of $T_{\text{sync}} = 20$ milliseconds is used, then, for a team of 10 units a consensus process would take around 2 seconds to complete. The adequacy of this value has to be

considered with respect to a specific application. As for the overhead introduced by the m_{sync} message, one needs first to determine its size. Recall that m_{sync} includes the following data structures: CRT, M, A, and clk. As an indicative example, consider that the team currently involves 10 nodes and the total number of messages in the CRT is 15. For each message there is an identifier, length, period, deadline, and offset, all expressed in slots, for example, using 1 byte for the first two and 2 bytes for the remaining ones. This means the CRT takes 120 bytes. The M matrix is a 10×10 bit structure, requiring 13 bytes. The A vector takes 10 bits, requiring 2 bytes. The clk includes a 4-byte slot counter plus an 8-byte representation of continuous time. There is still a timestamp (slot counter) associated with the CRT, taking extra 4 bytes. Altogether, this results in 151 bytes. Using the message payload of 106 bytes per slot as suggested above, the synchronization data takes 2 slots. With a $T_{\text{sync}} = 20$ milliseconds, this means 10% of the bandwidth dedicated to the synchronization mechanisms. If this is excessive in the scope of an application, then it is necessary to search for a better compromise between the overhead and the reactivity of the system, for example, increasing T_{sync}.

In order to facilitate the deployment of our protocol, it is also possible to build it on top of IEEE 802.11 for example. This allows taking advantage of the framing of the PDUs as well as of the respective physical layer. In this case, broadcast frames should be used, which are single frames transmitted without acknowledgment or RTS/CTS channel reservation. Indeed, these features are not needed as our protocol enforces colision-free transmission by means of sychronization and the bandwidth is reserved by the EDF scheduler. When compared to 802.11 DCF communication, our protocol requires the additional transmission of the m_{sync} message, which adds to its overhead. The use of fixed frame size also has a potential to increase the overhead of our protocol, requiring the transmission of several frames to convey an amount of data that would fit within a single standard 802.11 frame. The guarding windows, which depend on the achievable precision of the clock synchronization, may also contribute to insert extra idle time between the transmission of consecutive frames in our protocol. On the other hand, we are able to provide real-time communication, while 802.11 DCF is not, particularly with high-bandwidth utilization levels.

9. 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 excessive 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 system 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.

Our approach also integrates a connectivity tracking system that is proposed to support the resource reservation phase. It can also be used as basic component for an efficient packet routing strategy on top of the proposed MAC level communication protocol.

The protocol is meant for small sets of mobile units, typically between 10 and 20. However, it can be integrated into a hierarchical scalable routing framework, 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.

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 connectivity 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] Z. J. Haas, J. Deng, B. Liang, P. Papadimitratos, and S. Sajama, "Wireless ad hoc networks," in *Wiley Encyclopedia of Telecommunications*, J. G. Proakis, Ed., John Wiley & Sons, New York, NY, USA, December 2002.

[2] C. E. Perkins, *Ad Hoc Networking: an Introduction*, Addison-Wesley, Boston, Mass, USA, 2001.

[3] 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*, vol. 8, no. 3, pp. 293–308, 2000.

[4] J. Wu and I. Stojmenovic, "Ad hoc networks," *IEEE Computer*, vol. 37, no. 2, pp. 29–31, 2004.

[5] J. A. Stankovic, T. E. Abdelzaher, C. Lu, L. Sha, and J. C. Hou, "Real-time communication and coordination in embedded sensor networks," *Proc. IEEE*, vol. 91, no. 7, pp. 1002–1022, 2003.

[6] J.-D. Decotignie, "Wireless fieldbusses—a survey of issues and solutions," in *Proc. 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002.

[7] B. Hughes and V. Cahill, "Achieving real-time guarantees in mobile ad hoc wireless networks," in *Proc. Work-in-Progress Session of 24th IEEE Real-Time Systems Symposium (RTSS '03)*, pp. 37–40, Cancun, Mexico, December 2003.

[8] M. Gerharz, C. de Waal, M. Frank, and P. Martini, "Link stability in mobile wireless ad hoc networks," in *Proc. 27th Annual IEEE Conference on Local Computer Networks (LCN '02)*, pp. 30–39, Tampa, Fla, USA, November 2002.

[9] T. Srinidhi, G. Sridhar, and V. Sridhar, "Topology management in ad hoc mobile wireless networks," in *Proc. Work-in-Progress Session of 24th IEEE Real-Time Systems Symposium (RTSS '03)*, pp. 29–32, Cancun, Mexico, December 2003.

[10] S. H. Shah, K. Chen, and K. Nahrstedt, "Dynamic bandwidth management in single-hop ad hoc wireless networks," *Mobile Networks and Applications*, vol. 10, no. 1-2, pp. 199–217, 2005.

[11] N. H. Vaidya, P. Bahl, and S. Gupta, "Distributed fair scheduling in a wireless LAN," in *Proc. 6th Annual ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pp. 167–178, Boston, Mass, USA, August 2000.

[12] N. Johansson, U. Körner, and P. Johansson, "Performance evaluation of scheduling algorithms for bluetooth," in *Broadband Communications: Convergence of Network Technologies*, H. K. T. Danny and J. K. Paul, Eds., Kluwer Academic, Hong Kong, China, pp. 139–150, November 1999.

[13] M. Caccamo, L. Y. Zhang, L. Sha, and G. Buttazzo, "An implicit prioritized access protocol for wireless sensor networks," in *Proc. 23rd IEEE Real-Time Systems Symposium (RTSS '02)*, pp. 39–48, Austin,Tex, USA, December 2002.

[14] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[15] D. Dolev, C. Dwork, and L. Stockmeyer, "On the minimal synchronism needed for distributed consensus," *Journal of the ACM*, vol. 34, no. 1, pp. 77–97, 1987.

[16] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[17] J. Turek and D. Shasha, "The many faces of consensus in distributed systems," *IEEE Computer*, vol. 25, no. 6, pp. 8–17, 1992.

[18] X. Defago, A. Schiper, and P. Urban, "Total order broadcast and multicast algorithms: taxonomy and survey," Research Rep. IS-RR-2003-009, Japan Advanced Institute of Science and Technology, Ishikawa, Japan, September 2003.

[19] T. Facchinetti, G. Buttazzo, M. Caccamo, and L. Almeida, "Wireless real-time communication protocol for cooperating mobile units," in *Proc. 2nd International Workshop on Real-Time LANs in the Internet Age (RTLIA '03)*, Porto, Portugal, July 2003.

[20] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[21] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo, *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*, Kluwer Academic, Boston, Mass, USA, 1998.

[22] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass, USA, 1983.

[23] B. Liang and Z. J. Haas, "Predictive distance-based mobility management for PCS networks," in *Proc. 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99)*, vol. 3, pp. 1377–1384, New York, NY, USA, March 1999.

[24] C. Bettstetter, "Smooth is better than sharp: a random mobility model for simulation of wireless networks," in *Proc. 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '01)*, pp. 19–27, Rome, Italy, July 2001.

[25] C. Bettstetter and C. Wagner, "The spatial node distribution of the random waypoint mobility model," in *Proc. 1st German Workshop on Mobile Ad-Hoc Networks (WMAN '02)*, pp. 41–58, Ulm, Germany, March 2002.

Tullio Facchinetti is a 2001 computer science graduate from the Informatic Engineering Department, University of Pavia (Italy). He is currently waiting to defend his Ph.D. thesis, which was cotutored by Universities of Pavia (Italy) and Aveiro (Portugal), where he spent 6 months in 2004 researching on distributed wireless communication algorithms. During his Ph.D., he also worked at national and international projects funded by the Italian Ministry of University and Research and by the European Community. His main research interests are on real-time wireless communication, distributed systems, sensor networks, mobile units coordination, and neural networks.

Giorgio Buttazzo is an Associate Professor of computer engineering at the University of Pavia, Italy. He graduated with a major in in electronic engineering at the University of Pisa in 1985, received a Masters degree in computer science from the University of Pennsylvania in 1987, and a Ph.D. degree in computer engineering from the Scuola Superiore S. Anna of Pisa in 1991. During 1987, he worked on active perception and real-time control at the GRASP Laboratory, University of Pennsylvania, Philadelphia. From 1991 to 1998, he held a position of Assistant Professor at the Scuola Superiore S. Anna of Pisa, doing research on robot control systems and real-time scheduling. His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks. He is a Senior Member of the IEEE and the IEEE Computer Society.

Luis Almeida is an Assistant Professor at the Department of Electronics and Telecommunications, University of Aveiro, Portugal, since 1999. He is also a Senior Researcher at the IEETA research unit of the same university. Formerly, he was a design engineer in a company producing digital telecommunications equipment. He received a degree in electronics and telecommunications engineering in 1988 and a Ph.D. degree in electrical engineering in 1999, both from the University of Aveiro. His research interests lie in the fields of real-time networks for distributed industrial/embedded systems and navigation control for mobile robots.