Efficient Distributed Communications for Multi-robot Systems

João C.G. Reis[1], Pedro U. Lima[1], and João Garcia[2]

[1] Instituto de Sistemas e Robótica, Instituto Superior Técnico, Portugal
{jreis,pal}@isr.ist.utl.pt
[2] INESC-ID Lisboa, Instituto Superior Técnico, Portugal
jog@gsd.inesc-id.pt

Abstract. Wireless communications are one of the technical problems that must be addressed by cooperative robot teams. The wireless medium often becomes heavily loaded and the robots may take too long to successfully transmit information, resulting in outdated shared data or failures in cooperative behaviors that require synchronization among teammates. This paper introduces a novel solution to enable the immediate transmission of synchronization data in a way designed to reduce and better tolerate packet loss. It does so by categorizing the communications in multi-robot systems in two classes, robot state diffusion and synchronization messages. For the former, an existing adaptive transmission method (RA-TDMA) is used, and for the latter a novel solution was developed. Experiments show an important delay reduction when sending synchronization messages over a loaded network.

1 Introduction

One of the challenges in developing multi-robot systems lies with the communications among them. Wireless networks used without special structure or restrictions provide adequate communication facilities. However, in the face of restrictions or high load, it is necessary to use the network carefully to better exploit its capabilities without overloading it.

An example scenario of multi-robot communication is a robot soccer game, where this work was applied. The *Robot World Cup (RoboCup)*[5] was proposed in 1997 as an attempt to foster AI and intelligent robotics research. The *RoboCup Middle Size League (MSL)* is a senior competition where two teams of five robots play a soccer game. The rules used are a subset of the official *FIFA* Laws with added constraints on the robots and environment[8]. During games, robots communicate using only *IEEE 802.11a* or *IEEE 802.11b* network modes. Unicast and multicast communication modes are allowed and broadcast is forbidden. The maximum transmission bit rate allowed per team is 20% of *IEEE 802.11b* (2.2 Mbps). However, the rules are mostly not enforced in practice and several problems occur frequently, degrading communications quality.

Efficient communications are a key factor for the success of teams during competitions but most teams simply schedule their robots to transmit information periodically, without any synchronization among robots. Thus, in the worst

S. Behnke et al. (Eds.): RoboCup 2013, LNAI 8371, pp. 280–291, 2014.

case situation, all robots might try to transmit at the same time. Under such a situation, some packets would be transmitted before others, causing communication delays. Furthermore, collisions are very likely to occur, further delaying communication.

Some works show how identifying communication patterns is an important step in developing reusable software[12]. It is possible to distinguish two categories regarding communications in cooperative multi-robot systems. On the one hand *Robot State Diffusion* has to do with the robot's perception of the surrounding world using its own sensors (own position, the ball position the positions of obstacles). The robots continuously exchange this data to improve each robot's knowledge about the world. On the other hand, *Synchronization Messages* are related to the robots need to communicate to agree and keep relational behaviors[3,6] synchronized.

In this paper, a solution for multi-robot wireless communication is presented. This solution takes advantage of the communication patterns observed in cooperative multi-robot systems to better exploit the medium capabilities, thus enhancing a previous solution for periodic data transmission[10] with a mechanism to enable agents to quickly and reliably communicate with each other or in groups. This work is currently integrated into the *Robot Operating System (ROS)* middleware[9] and published as open source software[1].

Some good solutions for communications in multi-robot systems can be found in the literature. However, little attention is given to the types of communication required, as described above. The *Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture (CAMBADA)* team[2] of the University of Aveiro, Portugal, uses a middleware infrastructure specifically developed for their team. One of the components implements a *Reconfigurable and Adaptive Time Division Multiple Access (RA-TDMA)* communication protocol[11] with self-configuration capabilities to dynamically adapt to the number of active team members[10]. It works in a fully distributed way and has proved to be effective in game situations. There is an implementation available as free software[2]. This protocol, in which the solution presented in this paper is based, is adequate for robot state diffusion, but lacks explicit support for synchronization messages, for which our work provides a novel solution.

A communication system centered around a *message dispatcher*[13] is used by the RoboCup MSL *1. RFC Stuttgart* team[4] from the University of Stuttgart, Germany. In this solution, all agents establish a TCP connection to a central message dispatcher, a special entity in the system that filters messages according to time and other defined constraints. Messages with high priority are always sent; other messages are deleted if they become older than a specified threshold. In this solution, robot state is transmitted together with synchronization messages and the medium access can be affected by transmissions of outdated information. The priority mechanism transmits synchronization messages first, but messages must be transmitted to the dispatcher before this mechanism is

[1] SocRob Multicast, `http://www.ros.org/wiki/socrob_multicast`
[2] rtdb - Real-Time DataBase Middleware, `http://code.google.com/p/rtdb/`

used, adding an extra level of indirection. The message dispatcher also transmits all messages directly to all destinations, instead of using the multicast channel. This consumes bandwidth that could be used to transmit robot state information more frequently, thus keeping it more updated.

ROS[9] is a robotic middleware solution that has become widely used because of its quality of design and implementation. A *ROS* system is organized as a collection of nodes that communicate using messages. Nodes transmit and receive messages through two procedures: a publish-subscribe mechanism, named *topics*, where multiple nodes can publish messages and multiple nodes can subscribe to receive messages, and a direct node to node communication mechanism named *services*. However, there is no organized multicast solution that takes advantage of the characteristics of multi-robot systems to enhance communication. Furthermore, bandwidth limitation is not possible.

Transmission of real-time traffic in wireless networks is an interesting problem for many domains, e.g., factory automation. In [7], the authors propose a solution that can be implemented in *IEEE 802.11a/b/g* networks. However, it requires that some parameters of the network card are adjusted to give real-time stations priority over other stations. *E-MAC*[1] is a similar solution that avoids starvation of non real-time traffic. These solutions are not applicable to *RoboCup MSL* because of the need for modifying network card parameters, which although it is not explicitly forbidden in the rules, it would give an unfair advantage against teams not using it. Additionally if two teams would use this kind of solutions, conflicts would occur, unless the two teams cooperate somehow to arbitrate medium access. This is currently not allowed by the rules.

The rest of this document is organized as follows. Section 2 presents a detailed description of the *RA-TDMA* protocol. Section 3 describes in detail the proposed solution. Section 4 shows how the solution is integrated in the system. Section 5 presents the evaluation conducted on the solution. The conclusions are presented in section 6.

2 Reconfigurable and Adaptive TDMA

The RA-TDMA protocol[11] tries to disperse transmissions of all team members in time to avoid collisions within the team as much as possible, since the remaining network load cannot be controlled. Time is divided in slots of duration T_{tup} (team update period) in which all team members transmit once. T_{tup} is a configuration parameter set prior to execution and determines the global responsiveness of the system. Each of these slots is equally subdivided in slots for each active team member of duration T_{xwin}. Agents transmit at the beginning of their respective slots, thus spacing the transmissions as much as possible. Each agent uses only a fraction of its slot, the remaining time is used to accommodate delays in transmission and give the other team a chance to transmit.

This protocol does not need clock synchronization. After its own transmission slot, each agent keeps registering the exact time of arrival of its teammates' packets. The reception delay (δ) is calculated with respect to the expected time

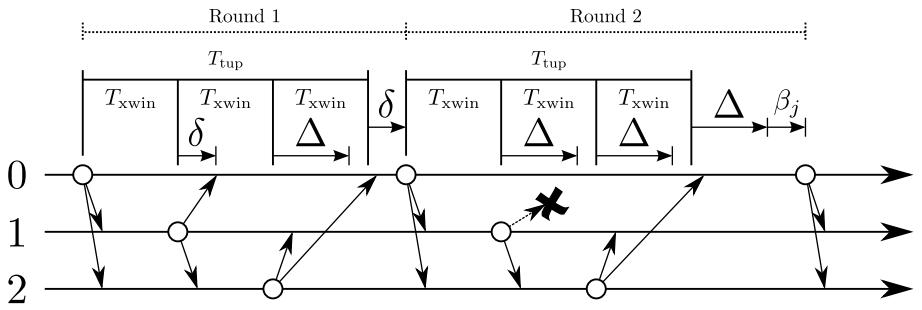


Fig. 1. Time diagram of two example rounds by a team of three agents

of arrival, which corresponds to the beginning of the transmitting agent own slot. The current round period is enlarged by the greatest of these delays. Only delays up to Δ are considered, with Δ being a global configuration parameter. Therefore, the effective round period will vary in the interval $[T_{\text{tup}}, T_{\text{tup}} + \Delta]$. An example situation is depicted in the first round of Fig. 1. At agent 0, the packet from agent 1 was expected after T_{xwin} but is received with a slight delay of δ. However, the packet from agent 2 is received with a delay greater than Δ, thus is ignored. The next packet transmitted by agent 0 is delayed by δ, because it was the only delay shorter than Δ.

When a robot does not receive any packet with a delay below Δ in a round, the next transmission will be delayed by a further β_j, different for every robot j. In this situation, the effective round period will be $T_{\text{tup}} + \Delta + \beta_j$. This is used to prevent situations in which the robots all keep transmitting but are unsynchronized. Having different round times in this situation will force the robots to resynchronize, since after a few rounds the robots will again receive a packet with a delay below Δ. An example is presented in the second round of Fig. 1, where the packet from agent 1 is lost and the packet from agent 2 is received with a delay greater than Δ.

Dynamic Reconfiguration. While in operation, the robots divide the *TDMA* round period by the number of active robots[10]. Since the robots can come in and out of play and malfunction during games, the number of active robots must be determined dynamically. Consequently, the transmissions are always separated as much as possible leaving no unused slots. Agents have two identifications: static *IDs* (SIDs) are given to each agent prior to execution within a pre-defined and known interval, and are used for agent identification. Dynamic *IDs* (DIDs) are used to identify only active teammates locally at each agent and are never transmitted. DIDs are assigned to each active agent, sorted by their SIDs. Each agent maintains a membership vector for all possible agents, indexed by their SIDs. Each agent may be in one of four states: *not running*, *insert*, *running* and *remove*. This vector is shared with the team by adding it to every transmission.

When an agent starts communicating, it sets its own state to *insert*. Agents that receive these initial packets set their state for the new agent as *insert*. When an agent X that has agent Y in *insert* state and detects that all agents in state *running* have agent Y in *insert* or *running* states, it updates the state of agent Y to *running*. The *DID* of agent Y is calculated and it is considered part of the *TDMA* round. Removing an agents follows a dual process. If nothing was received from that agent in the last rounds (the number of rounds is a configuration parameter), its state is changed to *remove*. When all agents have a given agent in *remove* or *not running* states, its state is changed to *not running*. The *DIDs* are reassigned and the round slots adjusted accordingly.

3 Solution Architecture

The *RA-TDMA* solution provides a suitable approach to dispersing robot state. However, it does not provide any special mechanism for transmission of synchronization messages. This is a problem for two reasons:

- Synchronization messages have to wait to be transmitted. This is a minimal delay for one single message but will accumulate in situations of especially bad network conditions where many packets fail to arrive correctly. This can be even worse if several rounds of communication are needed to synchronize the robots.
- There is no reception feedback. If the client software needs to be sure of correct reception, some mechanism must be implemented on top of the communication protocol. Again, this may take almost a full round time if no packets are lost, or much more with bad network conditions.

The solution proposed here can be seen as a protocol with two different modes of operation, each concerning one of the two robot communication patterns.

3.1 Robot State Diffusion

Robot state is transmitted using rounds of *Adaptive-TDMA* with a long period, divided by the number of active agents. To differentiate it from the transmission of synchronization messages, this mechanism is called *long rounds*. The long rounds closely are implemented using RA-TDMA, as described in section 2.

3.2 Synchronization Messages

When an agent needs to transmit synchronization information that is urgent or requires an answer as soon as possible, it initiates a new question round. Each question round is composed by the *question*, which is the initial data transmitted by the starting agent, and *answers*, that can be complex information or a simple acknowledgment. The mechanism to transmit synchronization messages is called *short rounds*, and might handle several question rounds simultaneously. In order

to easily distinguish short and long round packets, a different multicast socket is used. This socket uses the same multicast *IP* address that is used in the long rounds, but with a different port. This way, long and short round packets can be distinguished without increasing the packet size.

Basic Scenario. Each question round is identified by the SID of the agent that started it along with a *question identifier*, a number that uniquely identifies each question from a given agent. Every packet contains five elements:

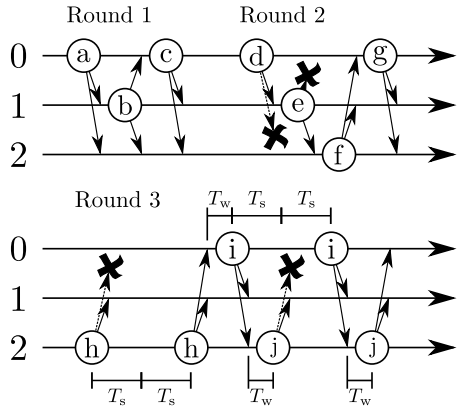
- The SID of the starting agent (SSID);
- The *question identifier* (QID);
- The *question* itself;
- A list of SIDs of agents that are *required* to answer;
- A list of *answers* along with the SIDs of the agents that produced each answer.

The initial packet transmitted by the starting agent contains the *question* and the list of SIDs of agents that are *required* to answer it. When this packet is received by another agent, it first verifies if its SID is in the *required* list. If this is the case, the agent will transmit a packet with its answer and its SID removed from the *required* list. Therefore, in a given packet regarding a specific question, each agent might be in one of three states:

- *Required*: The agent is required to answer this question and its SID is in the *required* list.
- *Answer*: An answer is present in the *answers* list of this packet.
- *Complete*: The answer was correctly received by the stating agent or no answer is required.

The question is only transmitted on packets that have any agent in the *required* state. Agents in the *answer* or *complete* states do not need the question, so if all agents are in either of these states the question field is left blank. When the starting agent has all the answers it requires, it will transmit a terminator packet with all agents in the *complete* state. This packet contains the two identifiers, but all other fields are left blank. With this last packet the question round is successfully completed. A simple example can be found in the first round of Fig. 2, where all packets are successfully received. The table shows the contents of the packets.

The proposed solution is flexible enough to accommodate various possibilities, an agent might want to transmit something to only one or to multiple agents. In any case, it is only necessary to add the proper SIDs to the *required* list. The proposed solution makes sure that all those agents receive the *question* and all their *answers* are returned to the starting agent, unless they are not reachable. Furthermore, what is important in a question round might be the initial data transmitted (*question*), the data that is transmitted by the agents in the *required* list, or both.



Packet	SSID	QID	Required	Answers
a	0	0	1	
b	0	0		1
c	0	0		
d	0	1	1, 2	
e	0	1	2	1
f	0	1		1, 2
g	0	1		
h	2	0	0	
i	2	0		0
j	2	0		

Fig. 2. Time diagram of three example short rounds by a team of three agents

All agents begin inactive. They become active as soon as a question round is started locally or a packet with a question round arrives. However, an agent will not become active if its SID is not in the *required* list of the received question. Note that several question rounds might be active at once. An agent might have started some, be required to answer some, and still there might be some others with which the agent is completely uninvolved. While an agent is active, it will transmit all it knows about all question rounds, even if they do not involve it. Only question rounds that involve it will keep the agent active.

Tolerance to Packet Loss. The protocol was designed to reduce and tolerate packet loss using as much redundancy as possible. When a packet is received with a question round started by some other agent, the agent will react to it if its SID is in the *required* list. It does not matter if the packet is the initial packet transmitted by the starting agent, or a packet already containing answers transmitted by some other agent. This way, if some agent does not receive the initial packet, it will likely get all the needed information from the next packet regarding that question round. Each agent also keeps saving all the answers from other agents. These answers will all be transmitted along with its own answer. This way, if some packet with an answer fails to reach the starting agent, the next packet will likely contain a copy of that answer. If an agent does not receive the final packet signaling completion, it will transmit its answer again. When the starting agent receives this, it will resend the terminator packet. The second round of Fig. 2 shows a situation where two packets are lost yet the round completes without any added delay.

Transmission Timing. Transmission of the initial packet determines the start of a question round. Agents transmit at the beginning of their own slots, which have a fixed duration of T_s. When an agent successfully receives a packet from

some other agent, it recalculates its next transmission time based on how many agents are active in the short rounds. Packet reception can only cause this time to be anticipated, to avoid situations where the transmission time is over delayed because of the reception of delayed packets.

When the received packet belongs to the agent that should have transmitted right before, the agent will ignore the slot duration and transmit much sooner. Transmission could happen immediately, but this would put an undesirable load on the medium since all agents might transmit in sequence without any interval. To avoid this, a short waiting time T_{w} is used between transmissions of successive agents, to create a window in which the medium is available to the other team. The slot duration and the short waiting time are configuration parameters. In the last round of Fig. 2 two packets were lost. Since only two agents are active, agents schedule retransmission with a period of $2 \times T_{\mathrm{s}}$. When a packet is received from the other agent, the next packet is transmitted after T_{w}.

Active Agents Estimation. For one agent to know if it is its turn to transmit, it must keep a record of which agents are active in the short rounds. It is not possible to know this for sure since packets might be lost or delayed, but it is possible to have an estimate that will be accurate in situations where all packets are promptly delivered. When this estimate fails, two or more agents might try to transmit simultaneously. This will increase the probability of packet loss. However, the waiting time between packets gives a better chance for these transmissions to succeed. The estimation is made based on the information that is kept about all active question rounds. Starter agents are always considered active, since the question round will only end after the terminator packet. All agents whose SID is in the *required* list of some active question round are also considered active.

3.3 Shared Concerns

Long and short rounds operate almost independently. Still, there is some information that must be shared between the two. Long rounds keep track of which agents are active at a given moment. This information is used by the short rounds to avoid waiting for transmissions from an agent that is not active. During operation, short rounds keep verifying if agents in the *required* list are in any state different from *not running*. This is done by the starting agent to ensure that the answers are delivered as soon as all the agents in the *required* list have answered or changed to the state *not running*. It is also done by all agents to keep the estimation of active agents as accurate as possible.

Short rounds are expected to be occasional and resolved quickly. Therefore, when the time comes to transmit in a long round, the agent will first check if there is a short round active. In this case, it will simply not transmit. Because short rounds may extend in time, this is only done once. After that, the agent will transmit anyway. This is necessary to resolve the case where an agent becomes inactive during a short round, and to guarantee that robot state keeps being diffused even in the presence of a heavy load of short rounds.

Bandwidth Management. It is easy to estimate how much bandwidth is needed by the long rounds based on the team update period T_{tup} and the size of the information that is transmitted. However, for the short rounds bandwidth usage is much harder to estimate because it depends on events in the game that will trigger question rounds and on packet loss because it uses retransmission. Packet sizes and the T_{tup} must be kept reasonable. Some attention must also be spent on ensuring that there are no situations that trigger an excess of question rounds. Still, a mechanism to ensure that the allowed bandwidth is not exceeded is necessary.

The bandwidth manager keeps track of a moment in time that represents the end of the last authorized transmission at the maximum allowed bandwidth. When an authorization request is received, if this moment is in the future, the request is immediately denied. If this moment is in the past, the request is accepted and the moment is then updated with the greatest of two moments: the current time or the sum of the previous moment and the duration of the requested transmission at maximum allowed bandwidth.

4 Integration with ROS

This solution is implemented as a library called *SocRob Multicast*. This library depends only on the *Boost C++* Libraries[3] for socket programming, threads and time control and on ROS[9] for logging and serialization of messages. To make the bridge between this library and the rest of the system, a ROS node must be created with a structure loosely following the one represented in Fig. 3. This node is responsible to create and process long round messages and knows what information must be shared, and also converts *ROS* services into short rounds. The correspondence between robots and SIDs is done by this node and can not be automated since it completely depends on the domain. In *RoboCup MSL*, the SIDs are the robot number minus one, because robots are numbered starting in 1 and SIDs start in 0. Although it is designed to comply with *RoboCup MSL* rules, this library is domain independent, enabling a *ROS* system to use one master per agent. If the connection between agents is severed, the agents will remain functional.

5 Evaluation

The evaluation of the proposed solution is centered on how fast information can travel from one robot to another, under different network conditions. Since the protocol presented is divided in two modes, it makes sense to evaluate these modes separately. The tests were conducted using one laptop per agent, with a team of five agents, in order to simulate real game conditions. The T_{tup} used was 100 milliseconds, T_{s} was 5 milliseconds and T_{w} was 1 millisecond. The laptops were connected to a *IEEE 802.11a* network. Each test was run for about

[3] Boost C++ Libraries, `http://www.boost.org/`

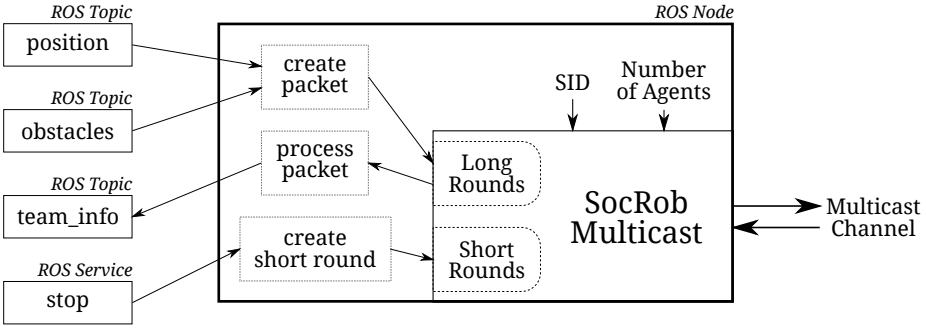


Fig. 3. Integration with ROS

one minute. The agent with SID 0 played the special role of initiating all question rounds and saving test results. The laptop clocks were synchronized using *chrony*[4]. During the tests, the greatest time difference reported by *chrony* was almost 2 milliseconds. Therefore, all results that depend on clock synchronization can be affected by this clock skew. All tests were run with and without load on the network. It is not possible to guarantee that the wireless medium is completely clear, thus when analyzing results with no network load it is necessary to consider that some interference is still possible.

5.1 Evaluation of Robot State Diffusion

CAMBADA team already proved that using *Adaptive-TDMA* is a better solution than a simple periodic transmission[11]. To prove that the same happens with our implementation of the RA-TDMA protocol, we created a periodic transmission solution that tries to transmit 10 times per second at a well defined moment. Thus, depending on the accuracy of time synchronization between computers, the agents will all try to transmit very close in time, simulating the worst case for periodic transmissions. Without network load, this degraded solution loses 5.629% of the packets, while our implementation of RA-TDMA loses almost no packet: only 0.038% of the total. In a loaded network, the degraded solution loses 7.981% of the packets, while RA-TDMA looses only 5.465%.

5.2 Evaluation of Synchronization Messages

Synchronization messages should be evaluated by the time it takes for all the receivers to get the question (question delay), and how long it takes for the initial agent to receive all the answers (answer delay). The accuracy of the question delay values depends on clock synchronization, since timestamps are taken in different computers.

[4] Chrony Home, `http://chrony.tuxfamily.org/`

Table 1. Synchronization messages delay. Time values are in milliseconds.

Network load:		Without		With	
Transmit using:		Long Rounds	Short Rounds	Long Rounds	Short Rounds
Question Delay	Minimum	100.487	1.098	101.934	1.374
	Average	116.488	2.407	136.935	22.227
	Maximum	431.995	51.470	948.631	358.010
Answer Delay	Minimum	145.564	6.976	177.353	13.368
	Average	214.030	12.044	247.494	69.924
	Maximum	509.755	102.012	1256.729	396.034

Four scenarios were considered: sending synchronization messages together with robot state using long rounds and using short rounds, both with and without network load. For each scenario, two experiments were made. In one experiment, agent with SID 0 sends a message to all other agents simultaneously. The results for this experiment can be found in Table 1. In another experiment, messages are sent to a single agent in turn. These results are not shown as they show similar, yet slightly smaller values. To simulate the worst case scenario when transmitting in the long rounds, the synchronization messages were started only in the moment after a robot state message was transmitted. Therefore, they had to wait a full round to be transmitted, hence the difference to the best case scenario is about 100 milliseconds in every value. The times achieved using short rounds are much smaller both with and without network load. The short rounds are particularly effective in avoiding extreme values, as the maximum value observed during the experiments was much smaller.

The experimental results are as expected, the short rounds bring a great advantage. The average delays are greatly reduced in all scenarios. Furthermore, the short rounds are effective in keeping the maximum values much lower.

6 Conclusions

The solution presented in this paper greatly improves the transmission of synchronization messages in the described scenarios. Transmission of robot state can be efficiently done using *Adaptive-TDMA*. However, waiting for the *Adaptive-TDMA* transmission slot can cause a great delay. This delay can have a great impact in robot performance, especially in situations where many packets are lost or more that one round of communication is needed to reach a decision. Middleware for robotics has seen a great evolution in recent years, with stabilization of good solutions like *ROS* that are now used in many robotic applications with evident advantages. However, the better known solutions lack support of advanced communication protocols like the one presented here. The proposed solution is a step further in the communication capabilities of multi-robot systems. It can even be used in many scenarios, including the *RoboCup MSL*, or if a team of robots needs to be deployed in a situation where a public wireless network must be used.

Acknowledgments. The authors would like to thank José Carlos Castillo Montoya for his comments and careful review of this paper.

References

1. Aad, I., Hofmann, P., Loyola, L., Riaz, F., Widmer, J.: E-MAC: Self-Organizing 802.11-Compatible MAC with Elastic Real-time Scheduling. In: IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems 2007, pp. 1–10. IEEE (2007)
2. Dias, R., Neves, A.J.R., Azevedo, J.L., Cunha, B., Cunha, J., Dias, P., Domingos, A., Ferreira, L., Fonseca, P., Lau, N., Pedrosa, E., Pereira, A., Serra, R., Silva, J., Soares, P., Trifan, A.: CAMBADA 2013: Team Description Paper (2013)
3. Drogoul, A., Collinot, A.: Applying an Agent-Oriented Methodology to the Design of Artificial Organizations: A Case Study in Robotic Soccer. Autonomous Agents and Multi-Agent Systems Journal 1(1), 113–129 (1998)
4. Käppeler, U.P., Zweigle, O., Rajaie, H., Häussermann, K., Tamke, A., Koch, A., Eckstein, B., Aichele, F., DiMarco, D., Berthelot, A., Walter, T., Levi, P.: 1. RFC Stuttgart Team Description 2010 (2010)
5. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: Robocup: The robot world cup initiative. In: Proceedings of the First International Conference on Autonomous Agents, pp. 340–347. ACM (1997)
6. Lima, P., Custódio, L.: Artificial Intelligence and Systems Theory Applied to Cooperative Robots. International Journal of Advanced Robotic Systems (3) (September 2004)
7. Moraes, R., Vasques, F., Portugal, P.: A TDMA-based mechanism to enforce real-time behavior in WiFi networks. In: IEEE International Workshop on Factory Communication Systems, WFCS 2008, pp. 109–112. IEEE (2008)
8. MSL Technical Committee: Middle Size Robot League Rules and Regulations for 2013, version - 16.1 20121208 (January 2013)
9. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software, vol. 3 (2009)
10. Santos, F., Almeida, L., Lopes, L.: Self-configuration of an Adaptive TDMA wireless communication protocol for teams of mobile robots. In: IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008, pp. 1197–1204. IEEE (2008)
11. Santos, F., Almeida, L., Pedreiras, P., Lopes, L., Facchinetti, T.: An Adaptive TDMA Protocol for Soft Real-Time Wireless Communication among Mobile Autonomous Agents. In: Proc. of the Int. Workshop on Architecture for Cooperative Embedded Real-Time Systems, WACERTS, vol. 2004, pp. 657–665 (2004)
12. Schlegel, C.: Communication patterns as key towards component-based robotics. International Journal of Advanced Robotic Systems 3(1), 49–54 (2006)
13. Zweigle, O., Kappeler, U., Haussermann, K., Levi, P.: Event based distributed real-time communication architecture for multi-agent systems. In: 2010 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), pp. 503–510. IEEE (2010)