Distributed Communications System for Multi-Robot Systems

João Carlos Galvão dos Reis

Dissertation for obtaining the Master Degree in
Information Systems and Computer Engineering

Jury
President: Professor José Carlos Alves Pereira Monteiro
Advisor: Professor Pedro Manuel Urbano de Almeida Lima
Co-advisor: Professor João Coelho Garcia
Evaluation jury: Professor Carlos Manuel Ribeiro Almeida

October 2012
Acknowledgments

I would like to thank Professor Pedro Lima for giving me the opportunity to work on the SocRob project. And as my advisor, along with Professor João Garcia, for all the support and help during this thesis.

I would also like to thank my parents, José and Luisa Reis, for all the support they’ve given me, with much more than words, during all these years. My grandparents for their permanent concern. My sister Rita, who is always close. And my girlfriend Nélia for all her love, patience and support.

I would like to thank all my friends. Those at the SocRob project, current and past, from whom I’ve learned so much. My friends at the Mobile Robotics Laboratory and 6.15. My classmates from Information Systems and Computer Engineering. And to all my friends who share with me the better parts of life.

Finally, a very special word for my godfather Raul Galvão. He always gave me strength and support, continuously pushed me forward and inspired me to do better. He always gave me all that he could and kept me focused on my objectives. Although he did not live to see it completed, I would like to dedicate this thesis to him.
Abstract

Wireless communications are one of the technical problems that must be addressed by cooperative robot teams and in particular by the competing teams in the Middle Size League of RoboCup. Data to be transmitted falls into two categories: robot state and synchronization messages. 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medium often becomes heavily loaded and the robots may take too long to successfully transmit information, resulting in outdated shared data or failures in relational behaviors. For the transmission of robot state, Adaptive-TDMA already provides a good solution. However, to transmit synchronization data, a novel solution is presented. Until now, these messages were transmitted together with robot state. This solution enables the immediate transmission of synchronization data in a way designed to reduce and better tolerate packet loss. Experiments where a message had to wait a full round to be transmitted yielded an average of 247 ms from the beginning to the reception of answers from all agents. With the solution presented here, the initial waiting time never happens and intermediate delays are greatly reduced, achieving the same behavior in 70 ms average. This results were obtained in a loaded network. In the same experiment done without network load and contacting only one agent, the time is reduced from 187 ms with the old solution to 10 ms with the one presented here.

Keywords

Network protocol; Wireless Communications; RoboCup Middle Size League (MSL); Multi-Robot Systems; Synchronized Robot Task Execution; Distributed State Representation.
Resumo

As comunicações por wireless são um dos problemas técnicos que têm de ser resolvidos pelas equipas que participam na liga de Futebol Robótico Médio do RoboCup. Durante os jogos, os dados a ser transmitidos dividem-se em duas categorias: estado do robot e mensagens de sincronização. O meio wireless fica frequentemente muito carregado e os robots podem demorar demasiado a transmitir a informação. Isto faz com que a informação partilhada fique desatualizada e com que os comportamentos relacionais falhem. Para transmitir o estado do robot, Adaptive-TDMA é uma boa solução. No entanto, para transmitir dados de sincronização, uma nova solução é apresentada. Até agora, estas mensagens eram transmitidas juntamente com o estado do robot. Esta solução permite que a transmissão de dados de sincronização seja feita imediatamente e foi projetada para reduzir e tolerar melhor a perda de pacotes. Em experiências em que uma mensagem tinha de esperar uma ronda inteira para ser transmitida, obteve-se uma média de 247 ms desde o início até à receção da resposta de todos os agentes. Com a solução aqui apresentada, o tempo de espera inicial nunca acontece e os atrasos intermédios são bastante reduzidos, obtendo-se uma média de 70 ms. Estes resultados foram obtidos numa rede carregada. Nas mesmas experiências feitas sem carga na rede e contactando apenas um agente, o tempo é reduzido de 187 ms com a solução antiga para 10 ms com a solução aqui apresentada.

Palavras Chave

Protocolo de rede; Comunicações Wireless; Futebol Robótico Médio RoboCup; Sistemas Multi-Robot; Execução Sincronizada de Tarefas Robóticas; Representação Distribuída de Estado.
# Contents

1 Introduction
   1.1 Communication Patterns .................................................. 3
   1.2 Wireless Communications in RoboCup MSL ................................. 5
   1.3 Current Situation in the RoboCup MSL .................................... 6
   1.4 Objectives ........................................................................... 7
   1.5 Document Structure ............................................................ 8

2 Related Work
   2.1 Reconfigurable and Adaptive TDMA ....................................... 11
      2.1.1 TDMA Basic Structure .................................................... 11
      2.1.2 Adaptive TDMA ............................................................ 11
      2.1.3 Dynamic Reconfiguration ................................................. 12
   2.2 Message Dispatcher ............................................................ 14
   2.3 Other Solutions ..................................................................... 15
   2.4 Conclusions .......................................................................... 16

3 Solution Architecture
   3.1 Long Rounds .......................................................... 19
   3.2 Short Rounds ............................................................ 21
      3.2.1 Basic scenario ............................................................. 21
      3.2.2 Features ....................................................................... 22
      3.2.3 Tolerance to packet loss ................................................. 23
      3.2.4 Transmission timing ....................................................... 23
      3.2.5 Active agents estimation ................................................. 23
   3.3 Shared Concerns ............................................................... 27
      3.3.1 Activation in Long Rounds ............................................. 27
      3.3.2 Activation in Short Rounds .......................................... 27
      3.3.3 Bandwidth Management ................................................ 27
   3.4 System Integration ............................................................. 29

4 Evaluation
   4.1 Evaluation of Robot State Diffusion ...................................... 33
List of Figures

1.1 Communication Patterns .................................................. 3

2.1 Time diagram of transmissions by a team of three agents ............... 11
2.2 Time diagram of one round by a team of three agents .................. 12

3.1 Time diagram the three stages for sending a packet ..................... 19
3.2 Basic scenario for a question round ........................................ 22
3.3 Question round with lost packets .......................................... 25
3.4 Question round with lost packets .......................................... 26
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Robot State age using periodic transmissions, without network load</td>
<td>35</td>
</tr>
<tr>
<td>4.2</td>
<td>Robot State age using periodic transmissions, with network load</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>Robot State age using Adaptive-TDMA, without network load</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>Robot State age using Adaptive-TDMA, with network load</td>
<td>35</td>
</tr>
<tr>
<td>4.5</td>
<td>Synchronization messages delay when transmitted with robot state, without network load</td>
<td>38</td>
</tr>
<tr>
<td>4.6</td>
<td>Synchronization messages delay when transmitted with robot state, with network load</td>
<td>38</td>
</tr>
<tr>
<td>4.7</td>
<td>Synchronization messages delay when transmitted using short rounds, without network load</td>
<td>38</td>
</tr>
<tr>
<td>4.8</td>
<td>Synchronization messages delay when transmitted using short rounds, with network load</td>
<td>38</td>
</tr>
<tr>
<td>4.9</td>
<td>Robot State age with low load of short rounds, without network load</td>
<td>40</td>
</tr>
<tr>
<td>4.10</td>
<td>Robot State age with low load of short rounds, with network load</td>
<td>40</td>
</tr>
<tr>
<td>4.11</td>
<td>Robot State age with heavy load of short rounds, without network load</td>
<td>42</td>
</tr>
<tr>
<td>4.12</td>
<td>Robot State age with heavy load of short rounds, with network load</td>
<td>42</td>
</tr>
<tr>
<td>4.13</td>
<td>Synchronization Messages delay with heavy load of short rounds, without network load</td>
<td>42</td>
</tr>
<tr>
<td>4.14</td>
<td>Synchronization Messages delay with heavy load of short rounds, with network load</td>
<td>42</td>
</tr>
</tbody>
</table>
Introduction

Contents

1.1 Communication Patterns ........................................ 3
1.2 Wireless Communications in RoboCup MSL .................... 5
1.3 Current Situation in the RoboCup MSL ....................... 6
1.4 Objectives .................................................................. 7
1.5 Document Structure ................................................ 8
1. Introduction

Cooperative multi-robot systems provide solutions for problems that one robot alone cannot handle. One of the challenges in developing multi-robot systems lies on the communications among them. In many scenarios, wireless networks provide adequate communication facilities. However, in more demanding scenarios, it is necessary to use the network carefully to better exploit its capabilities without overloading it. An example scenario is a soccer game, where a team of robots challenges another team of robots, or humans, with well defined rules and goals.

The Robot World Cup (RoboCup) [1] was proposed in 1997 as an attempt to foster AI and intelligent robotics research. It consisted initially of three competitions: real robot, software and special skills competition. The event is held annually and has grown to accommodate many new leagues, divided in junior and senior leagues. There are also local and national events with the same rules held throughout the world.

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 many added constraints on the robots and environment [2]. The field size is reduced, the robots size and weight is limited and all sensors must be on the robots. There is one computer outside the field that connects the robots to the referee’s computer and can optionally be used to process information. The robots communicate among themselves and the external computer using an IEEE 802.11 wireless network.

The SocRob (Society of Robots, Soccer Robots) team from Institute for Systems and Robotics, Instituto Superior Técnico, Portugal, has been competing in RoboCup’s RoboCup MSL since 1998. The team currently uses five omnidirectional motion robots, equipped with a dioptic vision system, a compass and a kicking device [3]. Each robot carries a laptop to fulfill its computational needs.

In this master’s thesis, a communication system for the SocRob team was created. This system consists of an application level protocol and its implementation in the existing software. Its objective is to deliver the best performance for the communication patterns observed in soccer robots over the heavily rule constrained wireless medium.
1.1 Communication Patterns

Identifying communication patterns is an important step in developing reusable robotic software [4]. Communications among soccer robots fall essentially in two categories during games, as illustrated in figure 1.1.

- **Robot State Diffusion**
  
  While the robots are in operation, they perceive the surrounding world using their own sensors. The generated information includes the robot's own position, the ball position if it is close enough and visible and the positions of obstacles surrounding the robot. The robots continuously exchange this data to improve each robot's knowledge about the world.

  This information is continuous by nature, so small delays in transmission are acceptable. The more time sensitive information is that about the surroundings of the robot, which is likely perceived by the robot's own sensors. In any case, the data should be time-stamped so that outdated information can be discarded.

- **Commands and Relational Behaviors**
  
  The robots receive commands from the external computer which is connected to the referee’s. Furthermore, the robots need to communicate to agree and keep relational behaviors synchronized, as during ball passes.

  This messages should be transmitted as soon as possible and received by all the robots that need to receive them in a short amount of time. Delays might cause a robot to lose opportunities in the game or not to comply with the referee’s orders risking punishment.
1. Introduction

During robot development, it may be useful for the robots to transmit debug information or be manually controlled. However, this does not happen during games and is not relevant for the robots to attain their goals. Furthermore, debug information follows either of the two patterns described above. Therefore, the two patterns described above suffice for both development and actual game situations.

Debug data follows the same pattern as the robot state information. It consists mainly of unprocessed and intermediate sensor information and the robot's decisions at various levels. Manually controlling the robot can be accomplished in the same way as the transmission of referee commands.
1.2 Wireless Communications in RoboCup MSL

During games, robots communicate using an IEEE 802.11 network. However, the RoboCup MSL rules specify additional constraints on the utilization of the network. Only IEEE 802.11a and IEEE 802.11b modes are allowed. Teams choose which network mode to use, and both may use the same. Access points are provided by the organization and ad-hoc networks are explicitly forbidden. Unicast and multicast communication modes are allowed and broadcast is forbidden. Each team has pre-defined unicast and multicast IPv4 addresses. Bandwidth is limited to 20% of IEEE 802.11b per team, which is the slower mode. Therefore, the maximum bit rate is 2.2 Mbps.

Efficient communications are a key factor for the success of teams during competitions. However, the rules are mostly not enforced in practice and several problems occur frequently, degrading communications quality [5].

The main issue is network conflicts when playing against teams that do not respect the bandwidth limitation, causing excessive delays and packet loss. There is one team that frequently sends bursts of twelve 1.5 kB packets, occupying the medium for 10 ms (if using IEEE 802.11a, the faster mode), making it difficult for the other team to transmit during this time. Another team transmits almost continuously, using a total bandwidth of 13.75 Mbps, far beyond the 2.2 Mbps allowed.

Furthermore, the APs placement and channel assignment can be sub-optimal. Other devices can be using the same channel, like pre-installed wireless networks for internet access or the teams own APs installed nearby.
1. Introduction

1.3 Current Situation in the RoboCup MSL

Most teams simply schedule their robots to transmit information periodically, without any synchronization among robots. Thus, in the worst situation, robots might all try to transmit at the same time. In this situation, some packets will be transmitted before others, causing communication delays. Furthermore, collisions are very likely to occur, further delaying communication.

There is one solution that tries to organize transmissions of all robots in the team, in an attempt to mitigate collisions [6]. Using this solution, only collisions with packets from the opponent team or other networks are likely to occur. However, this solution does not take into account situations where a robot needs to transmit synchronization information. Furthermore, it does not provide any automatic way of knowing if transmitted data was successfully received. There is currently no known solution that simultaneously tries to avoid collisions and provide support for both communications patterns.
1.4 Objectives

The medium has to be shared as stated in the rules and the opponent team transmissions cannot be controlled. Therefore, the only thing that can be done is to control the team’s own transmissions. This is the case in RoboCup MSL as it is in many multi-robot systems. This work proposes a protocol that tries to separate transmissions from different robots as much as possible in time, in order to minimize collisions and interference within the team.

The novel contribution is to take into account the two communication patterns observed in cooperative multi-robot systems. State diffusion messages can be slightly delayed when the medium is heavily loaded to better accommodate alien traffic. However, robot commands have to be delivered as soon as possible.

Objectives: This work proposes a wireless communication protocol for multi-robot systems that combines the transmission of robot state and synchronization information so that robots can keep shared information updated and perform synchronization communications in a fast and reliable way. It should provide dependable communications in the presence of heavy network load.

Furthermore, the project has ported its code base to a new middleware, ROS[7]. With this, it benefits from off-the-shelf common robotic components in addition to an increased organization of its own internal structure.

The solution presented in this thesis is fully integrated in ROS. It is reusable and can be used even outside the scope of RoboCup MSL.
1. Introduction

1.5 Document Structure

The rest of this document is organized as follows. Chapter 2 will present other solutions to the same and similar problems. Chapter 3 will describe in detail the architecture of the proposed solution. Chapter 4 will present the evaluation conducted on the solution. This document closes up with the conclusions and possible future work on chapter 5.
Related Work

Contents

2.1 Reconfigurable and Adaptive TDMA ........................................... 11
2.2 Message Dispatcher ................................................................. 14
2.3 Other Solutions ............................................................... 15
2.4 Conclusions ................................................................. 16
2. Related Work

Most teams in RoboCup MSL use custom built communication systems developed over time to answer immediate necessities. Some teams, however, have paid special attention to this issue and developed solutions that provide good performance while respecting the rules.
2.1 Reconfigurable and Adaptive TDMA

The CAMBADA (Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture) team [8] of the University of Aveiro, Portugal, has been competing in the Middle Size League with good results, including RoboCup’s 2008 first place. They use a middleware infrastructure specifically developed for their team, composed of several components [9]. One of these components implements an adaptive time division multiple access (Adaptive-TDMA) communications protocol [10] with self-configuration capabilities to dynamically adapt to the number of active team members [6]. This protocol works in a fully distributed way and has proved to be effective in game situations. There is an implementation available as free software [11].

2.1.1 TDMA Basic Structure

This protocol 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 $T_{xwin}$.

At the beginning of their respective slots, the agents concatenate all items to be transmitted into the required number of packets (usually just one) and dispatch them to the medium. This way, the transmissions are separated 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. An example with three agents is given in figure 2.1.

2.1.2 Adaptive TDMA

The protocol tries to compensate heavy load on the network by enlarging the TDMA round period $T_{tup}$, thus consuming less bandwidth. This will hopefully reduce the stress on the network, allowing fewer but less delayed transmissions and reducing the number of lost packets.

After its own transmission slot, each agent keeps registering the exact time of arrival of its team mates’ packets. The reception delay ($\delta$) is calculated with respect to the expected time of arrival. The current round period is enlarged by the greatest of these delays. Only delays up to $\Delta$ are considered, with $\Delta$ being a global configuration parameter. Therefore, the effective round period will vary in the interval $[T_{tup}, T_{tup} + \Delta]$. An example situation is depicted in figure 2.2.
2. Related Work

Figure 2.2: Time diagram of one round by a team of three agents
This diagram shows one round, as perceived by agent 0. The reception of the packet from agent 1 is slightly delayed by $\delta$. However, the reception of the packet from agent 2 is delayed more than $\Delta$, thus is ignored. The next packet transmitted by agent 0 is delayed by $\delta$, because it was the only delay shorter than $\Delta$.

When a robot does not receive any packet with a delay below $\Delta$ in a round, the next transmission will be delayed by a further $\beta_j$, different for every robot $j$. In this situation, the effective round period will be $T_{tup} + \Delta + \beta_j$. This is used to prevent situations in which the robots all keep transmitting but unsynchronized. Having different round times in this situation will force the robots to resynchronize.

2.1.3 Dynamic Reconfiguration

While in operation, the robots divide the TDMA round period by the number of active robots. 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, in order to better accommodate uncontrolled traffic. Currently, $T_{tup}$ is kept constant, only $T_{xwin}$ is adjusted.

Agents have two identifications (IDs). Static IDs are give to each agent prior to execution within a pre-defined and known interval. Agents identify themselves by their static IDs. Dynamic IDs are used to identify only active team mates locally in each agent and are never transmitted. They are assigned to each active agent starting from 0, by order of their static IDs.

Each agent maintains a membership vector for all possible agents, indexed by their static IDs. 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.

The membership vector is initialized with all agents in the state not running. This state means that the agent is not communicating and has no slot in the TDMA round.

When an agent starts communicating, it sets its own state to insert, meaning it needs to be assigned a slot in the round. The agent starts transmitting in unsynchronized mode. The transmitted packets will compete for the medium with those of other team mates (if there are team mates already communicating) but should fit in the unused portions of their slots. Agents that receive these 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 dynamic ID is calculated and it is considered part of the TDMA round. Then the protocol enters in the scan mode, in which agents use a slightly longer period to find their new slots. The agent with dynamic ID 0 plays a special role in this mode, it is the only one that uses the normal update period and all others...
synchronize only with it, to avoid creating subgroups of synchronized agents.

Removing an agent follows a similar 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 dynamic IDs are reassigned and the round slots adjusted accordingly.
2. Related Work

2.2 Message Dispatcher

The RFC Stuttgart team [12] is a RoboCup MSL competitor from the University of Stuttgart, Germany. They won first place in RoboCup 2009. They use a communication system centered around a message dispatcher [13].

In this solution, all agents establish a TCP connection to a central message dispatcher, a special entity in the system. TCP was chosen because it provides guarantees of message delivery. However, in cases of bad network quality, it may perform worst than UDP because automatic retransmissions will further strain the medium. To mitigate this problem, a special implementation of the protocol is used, but its details are not specified.

The message dispatcher only filters messages according to time and other defined constraints. Processing data at this point is not desirable in a system where agents should be autonomous. Having a central message dispatcher has the advantage that agents do not need to manage direct connections to other agents. The only thing an agent needs is the address of the message dispatcher and manage the connection to it.

TCP provides facilities to transmit a continuous stream of data between two entities. As a consequence of this, messages must be converted either to pure binary or a text string and separation of individual data items must be done explicitly. One possibility is to send the size of the item before it and another is to use a special terminator symbol. This solution converts messages to strings and uses a special end marker and a line break as terminators. This was chosen because it facilitates debug and makes it possible to process messages while previous ones are still being transmitted, however, exactly how this can be done is not specified. Each message has a header which includes identification of message type, a timestamp, the priority of the message and an option to broadcast it or deliver only to specific agents. Only the header needs to be decoded by the message dispatcher, facilitating the forward procedure.

When connecting to the message dispatcher, clients send their identification and two options. About broadcast messages, clients can choose to receive only, send only or both. Clients can also specify if they are part of the team or just an external program.

The message dispatcher sorts received messages by priority. Messages with high priority are always sent, however, other messages will be deleted if they become older that a specified threshold. Connections may terminate abruptly, without the well defined termination sequence of TCP. Therefore, the message dispatcher closes connections that have been inactive for more that a defined amount of time. The agents follow a similar process to assure that the connection to the message dispatcher is still alive. When only one agent is connected, dummy messages are sent periodically to keep the connection alive.
2.3 Other Solutions

Transmission of real-time traffic in wireless networks is an interesting problem for many domains, e.g., factory automation. Several solutions have been proposed for the transmission of soft real-time traffic over IEEE 802.11 networks.

In [14], 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. Real-time stations are coordinated among themselves in a TDMA way to avoid conflicts. E-MAC[15] is a similar solution that avoids starvation of non real-time traffic.

These solutions are not applicable to RoboCup MSL for two reasons. First, while modifying network card parameters is not explicitly forbidden in the rules, it would give an unfair advantage against teams not using it. Thus it is against the rules philosophy. Second, if two teams would use this kind of solutions, conflicts would occur, unless the two teams cooperate somehow to decide medium access. This is currently not allowed by the rules.
2. Related Work

2.4 Conclusions

The communication solutions provided by most middleware solutions like ROS\[7\] do not leverage the characteristics of multi-robot systems. E. g., since participating agents are well known, it is possible to disperse the transmissions in time as Adaptive-TDMA does. ROS does not offer any mechanism to do this. Furthermore, ROS only offers the means to communicate within the same system. Thus, if a connection fails, the whole system fails.

The Reconfigurable and Adaptive TDMA solution provides a suitable approach to dispersing robot state. However, it does not provide any special mechanism for transmission of synchronization messages, they have to be transmitted together with state messages. This is a problem for two reasons:

1. Synchronization messages have to wait to be transmitted. This is a minimal delay for one single message. However, it will accumulate in situations of specially 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.

2. 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 using a central dispatcher uses TCP to ensure message delivery. However, robot state is also transmitted this way and the medium may be wasted with transmissions of outdated information. There is a priority mechanism to transmit synchronization messages first, but messages must be transmitted to the dispatcher before this mechanism is used. The message dispatcher is an extra level of indirection, thus the message must be transmitted twice to reach its destination. This is a small delay with good network conditions. However, it may become relevant if the medium degrades too much. 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.
3 Solution Architecture

Contents

3.1 Long Rounds ................................................................. 19
3.2 Short Rounds .............................................................. 21
3.3 Shared Concerns .......................................................... 27
3.4 System Integration ....................................................... 29
3. Solution Architecture

The proposed solution can be seen as a protocol with two different modes of operation, each concerning one of the two robot communication patterns identified in section 1.1. However, both co-operate in order to achieve the single objective of robot communication, providing a single integrated solution.

Two different modes are needed because when robot state messages are lost there is no need to detect the loss to retransmit it immediately. A message with newer information should be transmitted soon. Instead, the bandwidth is better used to accommodate external traffic or synchronization messages currently being transmitted. However, synchronization messages must be delivered as soon as possible, justifying retransmission.

Robot state diffusion is accomplished using Adaptive-TDMA, the solution developed by the CAMBADA team[6]. On the other hand, synchronization communications like commands or behavior synchronization messages immediately trigger transmission of data and acknowledgment or answer in return. This is done in a TDMA way with a very small period. Instead of enlarging the period in case of packet loss, the period is shortened in case of successful reception. Retransmissions and acknowledgments are used to guarantee message reception.

All packets are transmitted using UDP to two multicast groups composed of all the robots in the team. One of these groups is dedicated to robot state diffusion and the other to synchronization messages. They share the same IP address but use different ports.
3.1 Long Rounds

Robot state is transmitted using rounds of Adaptive-TDMA with a long period (in RoboCup MSL settings a possible value is about one tenth of a second). Robots have static and dynamic IDs and the long rounds are divided only by the number of active agents. When the network load is high and packets get delayed, the round period is enlarged. All of this is implemented as described in section 2.1.

The implementation of long rounds closely follows the solution of the CAMBADA team. However, several modifications were made:

- Sending a packet is clearly divided in three stages (figure 3.1):
  - Control: The exact moment for the transmission is decided after the maximum delay $\Delta$ for the last expected packet in a round. At this moment all the packet delays that are small enough to be considered are known.
  - Preparation: To send the most updated information possible, the packet to send must be prepared as close as possible to the transmission. The time that it takes to prepare a packet may vary, so an estimation is made. The time that was used to prepare the last few packets is kept in a circular buffer, and the maximum of these times multiplied by a constant is used as the estimation. The packet is prepared exactly this time before the moment it should be sent.
  - Send: The already prepared packet is sent. If the packet is still not ready, it will be sent as soon as possible, but the preparation time estimation should avoid this to happen frequently.
3. Solution Architecture

- The membership vector is no longer shared. The four states *not running, insert, running* and *remove* are still used, but only locally in each agent. Only a vector of boolean values is shared. This boolean value represents if an agent is *active* or not. States *insert* and *running* correspond to active agents and states *remove* and *not running* to inactive agents. This is enough information for the dynamic reconfiguration to function properly.

- To account for situations where the system is heavy loaded and there is not enough processing power to keep the protocol functioning properly, if the packet transmission is delayed for more than $T_{tup}$ then one full round is immediately skipped.

The implementation of long rounds also had to be adjusted for the existence of short rounds. Before sending a packet, it first checks if the short rounds are active. If so, the packet is not transmitted to leave more bandwidth available. However, in the next and following rounds, if short rounds are still active then it will transmit anyway. When short rounds are detected inactive, this mechanism will reset.

Short rounds are expected to end quickly, if not then it is probably because they are waiting for an answer from an agent which is no longer active. Long rounds are responsible to detect this situation and determine that the agent is not running.
3.2 Short Rounds

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. Short rounds might transmit 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, however the port is different. This way, long and short round packets can be distinguished without increasing the packet size.

3.2.1 Basic scenario

Each question round is identified by the static ID 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 static ID of the starting agent;
- The question identifier;
- The question itself;
- A list of static IDs of agents that are required to answer;
- A list of answers along with the static IDs of the agents that produced each answer.

The initial packet transmitted by the starting agent contains the question and the list of static IDs of agents that are required to answer it. When this packet is received by another agent, it first verifies if its static ID is in the required list. If this is the case, the agent will transmit a packet with its answer and its static ID 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 static ID 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 in packets that have any agent in state required. Agents in states answer or complete 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 required, it will transmit a packet with all agents in state complete. 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 is illustrated in figure 3.2.
3. Solution Architecture

Figure 3.2: Basic scenario for a question round
Agent with static ID 0 sends a message directly to agent with static ID 1. Other agents are either not communicating or do not need to receive this message. Agent 1 responds with its answer, and removes its SID from the required list. Agent 0 then finishes this question round by transmitting a packet with no required agents or answers.

(Key: SID - static ID; SSID - static ID of starting agent; QID - question ID.)

3.2.2 Features

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 static IDs 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.

In cases where the question is what is important, all the agents in the required list will receive it if they are reachable. Feedback of reception is provided to the starting agent as a list of the agents that successfully received the data. The protocol will only give up on contacting an agent when it is considered not running by the long rounds, in this case its static ID will not be part of the list. It is possible that an agent receives the question but packets with the acknowledgment are all lost in the network. This is not an issue since that agent is considered out of the team and will be treated as a new agent once it becomes reachable again.

If the answers are important, the protocol will make sure they are delivered to the starting agent, unless for agents which are out of the team. In fact, there is no real distinction between answers and acknowledgments, since these are simply answers with an empty data field.

All agents start inactive in the short rounds. 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 static ID 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 others and still there might be some others with which the agent is completely unconcerned. While an agent is active, it will transmit all it knows about for all question rounds, even if they do not concern it. However, only question rounds that concern it will keep the
agent active.

### 3.2.3 Tolerance to packet loss

The protocol was designed to reduce and tolerate packet loss. Packets are transmitted in a way that will hopefully reduce packet loss, which is described below in detail. Tolerance to packet loss is achieved by 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 static ID 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.

An example where two packets are lost is depicted in figure 3.3 on page 25.

### 3.2.4 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 transmit right before, the agent will ignore the slot duration and transmit much sooner. Transmission could happen immediately, however 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 wait time are configuration parameters.

Figure 3.4 on page 26 illustrates the retransmission after a packet loss.

### 3.2.5 Active agents estimation

For one agent to know if it is the next one to transmit, it must keep an estimation 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 estimation that will be accurate in situations where all packets are promptly delivered. When this estimation 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.
3. Solution Architecture

The estimation is done 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 static ID is in the *required* list of some active question round are also considered active.
Agent with static ID 0 sends a message to agents 1 and 2, however, it never reaches agent 2. Other agents are either not communicating or do not need to receive this message. Agent 1 responds with its answer, and removes its SID from the required list. This packet never reaches agent 0, only agent 2 receives. Therefore, this packet is the first one that agent 2 receives. It contains enough information for agent 2 to react, thus agent 2 is able to transmit its answer and retransmit the answer from agent 1 in its time. Agent 0 receives the two answers simultaneously. It then finishes this question round by transmitting a packet with no required agents or answers.

(Key: SID - static ID; SSID - static ID of starting agent; QID - question ID.)
3. Solution Architecture

Agent with static ID 0 sends a message directly to agent with static ID 1. Since two agents are active, the next transmission is scheduled to happen in $2 \times T_s$ time. Other agents are either not communicating or do not need to receive this message. Agent 1 responds with its answer, and removes its SID from the required list. However, agent 0 never receives this packet, and retransmits the first packet at the scheduled time. When this packet arrives at agent 1, it will shorten its waiting time to $T_w$, and retransmit the packet then. Agent 0 receives the packet and then finishes this question round by transmitting a packet with no required agents or answers. (Key: SID - static ID; SSID - static ID of starting agent; QID - question ID; NTS - next transmission schedule.)
3.3 Shared Concerns

Long and short rounds operate almost independently. However, there is still some information that must be shared between the two.

3.3.1 Activation in Long Rounds

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.

3.3.2 Activation in Short Rounds

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. However, 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.

3.3.3 Bandwidth Management

Bandwidth usage depends on several factors. 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. However, a mechanism to ensure that the allowed bandwidth is not exceeded is still necessary.

Both long and short round managers, immediately before transmitting, ask authorization from the bandwidth manager. If it is denied, they will simply not transmit. The effects will be the same as if the packet was lost.

The bandwidth manager was designed so that the maximum allowed bandwidth is respected, but authorization requests too close in time do not get unfairly denied. When requests are all properly spaced in time, all will get accepted. Transmission requests reserve a duration of time which corresponds to the transmission of data at maximum allowed bandwidth. Whenever possible, this duration is reserved in the past. However, if the last transmission was too close, this duration will be reserved partially in the future. Until this duration is over, all requests will get denied.
3. Solution Architecture

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;
- The sum of the previous moment and the duration of the requested transmission at maximum allowed bandwidth.
3.4 System Integration

This solution is implemented as a library called *SocRob Multicast*. This library depends only on the Boost C++ Libraries\textsuperscript{[16]} for socket programming, threads and time control and on ROS\textsuperscript{[7]} for logging and serialization of messages.

To make the bridge between this library and the rest of the system, a ROS node was created. This node interacts with the rest of the system through ROS topics and services. It is responsible to create and process long round messages and knows what information must be shared. It also converts ROS services into short rounds. This functionality could not be automated since ROS services support only a single target, and the short rounds support queries to more than one robot. Furthermore, the correspondence between robots and static IDs is done by this node. This correspondence could not be automated since it completely depends on the domain. In RoboCup MSL, the static IDs are the robot number less one, because robots are numbered starting in 1 and static IDs start in 0. In other domains, the static IDs have to be given based on some other domain characteristic.

Although it is designed to comply with RoboCup MSL rules, this library is domain independent, it is not restricted to be used in robotic soccer. It enables a ROS system to use one master per agent. Since all the nodes must be permanently connected to the master, this enables each agent to function independently from the others. If the connection between agents is severed, the agents will remain functional. This solution provides support for robot state diffusion and synchronization messages. The former is similar to ROS topics in an unreliable transport, and the latter to ROS services. To provide reliable transmission of ROS topics, synchronization messages should be used. Furthermore, the support to query multiple agents is added.
3. Solution Architecture
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Evaluation of Robot State Diffusion</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Evaluation of Synchronization Messages</td>
<td>36</td>
</tr>
<tr>
<td>4.3</td>
<td>Evaluation of Interference</td>
<td>39</td>
</tr>
<tr>
<td>4.4</td>
<td>Evaluation of Degradation</td>
<td>41</td>
</tr>
</tbody>
</table>
4. Evaluation

The evaluation of the proposed solution will focus 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. However, the interference between the two must also be evaluated.

The tests were conducted using one laptop per agent, with a team of five agents, in order to simulate real game conditions. The laptops were connected to a IEEE 802.11a network, since it is the faster allowed and the one that is used in the games. Each test was run for about one minute. The agent with static ID 0 played the special role of initiating all question rounds and saving test results.

The laptop clocks were synchronized using chrony. During the tests, the greatest time difference reported by chrony was almost 2 milliseconds. Therefore, all results that depend on clock synchronization can be wrong by this amount.

All tests were run twice: with and without load on the network by an external source. 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. At least chrony had to use the network for clock synchronization, and external disturbances are also possible.

Network load was generated using two other laptops. Both these laptops transmitted and received random data simultaneously from the other, using UDP at maximum possible bandwidth.
4.1 Evaluation of Robot State Diffusion

Robot state consists of information that is updated several times per second by sensing algorithms. When the information is accessed in the robot that produced it, it is always the newest data available. However, to be accessed from other robots, the information must first be transmitted over the network. Thus, the communication protocol plays the important role of keeping information updated in all the robots.

Robot state diffusion is done using the long rounds. These use Adaptive-TDMA, the solution developed by the CAMBADA team. They already proved that using Adaptive-TDMA is a better solution than a simple periodic transmission, but only considering packet loss \[10\]. Therefore, the tests have the objective of proving that the same happens in this implementation and is indeed an advantage over the previous solution used by our team.

Evaluation of robot state diffusion should be based on how much outdated the information is. This depends on the sum of 3 time durations:

- How long the information is on the producing robot before the time comes to start transmission in a round;
- The transmission time itself;
- How long the information stays on the receiver without being replaced.

Knowing how old the information is when it arrives and how long it stays in use, it is possible to calculate a continuous average of how outdated the information is. This is the most relevant evaluation metric. The fraction of lost packets is also relevant to evaluate how successful the protocol is on avoiding packet loss. Results from the CAMBADA team only include information on packet loss, hence only this can be compared.

The old solution used by our team consists of periodic transmissions with a fixed time interval between them. Hence, the schedule for all transmissions depends only on when the first one is made. The moment of the first transmission is not determined, it happens some time after the software starts running. This may, on one extreme, lead to a situation where all agents try to transmit at the same moment or, on the other extreme, distribute transmissions perfectly in time.

Adaptive-TDMA is designed to avoid having multiple agents trying to access the medium at the same time. Consequently, it only brings advantage over a situation where periodic transmissions would all try to transmit at the same time.

Table 4.1 shows the result of one experiment using periodic transmissions. The software was changed to schedule the transmissions 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. Table 4.2 shows the result of the same experiment but with the network loaded with external traffic.

Tables 4.3 and 4.4 show the result of experiments using the developed solution, without and with network load.
4. Evaluation

These tables show the delay of information from other agents on agent with static ID 0. To test for the worst case, information is produced at the source immediately after the transmissions. In an ideal situation, an agent transmits 10 times per second and the transmission is instantaneous. In this scenario, at the receiving agents, the information will be exactly 100 milliseconds old when received and 200 milliseconds old when replaced. The average age in this situation is exactly 150 milliseconds. This is the minimum possible value.

Two delay sets are considered: ignoring and considering transmission delay. Times ignoring transmission delay consider that, when a packet arrives, the information it contains is as old as it was when the packet was transmitted. Hence, these values are certain because they do not depend on clock synchronization. To be able to consider transmission delay, clock synchronization is needed, and therefore a small error is expected in values that take into account transmission delay.

Average columns display the continuous average of how much information from a given agent is outdated. Maximum columns display the maximum age of information at time of replacement.

All packets carry a sequence number. Gaps in reception can be detected and this enables packet loss to be calculated.

It is immediately clear that Adaptive-TDMA is effective in reducing packet loss. For the scenarios with no network load, the average age of information displays the expected results. The average age of information is slightly less when using the new solution, which is justified by the reduced packet loss.

However, the results are not conclusive in the scenarios with network load. Note that Adaptive-TDMA enlarges the round period to better accommodate external traffic. Consequently, the presence of external traffic will increase the average age of robot state information. On the other hand, Adaptive-TDMA is still effective in keeping the maximum age of information lower, especially if transmission time is considered.
4.1 Evaluation of Robot State Diffusion

<table>
<thead>
<tr>
<th>Agent</th>
<th>Packet loss</th>
<th>Ignoring transmission</th>
<th>Considering transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>3.730 %</td>
<td>155.128</td>
<td>330.332</td>
</tr>
<tr>
<td>2</td>
<td>7.593 %</td>
<td>167.832</td>
<td>517.431</td>
</tr>
<tr>
<td>3</td>
<td>4.304 %</td>
<td>168.399</td>
<td>510.061</td>
</tr>
<tr>
<td>4</td>
<td>6.887 %</td>
<td>158.342</td>
<td>350.454</td>
</tr>
</tbody>
</table>

Table 4.1: Robot State age using periodic transmissions, without network load
Time values are in milliseconds.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Packet loss</th>
<th>Ignoring transmission</th>
<th>Considering transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>8.397 %</td>
<td>161.223</td>
<td>507.059</td>
</tr>
<tr>
<td>2</td>
<td>8.563 %</td>
<td>160.654</td>
<td>456.923</td>
</tr>
<tr>
<td>3</td>
<td>8.550 %</td>
<td>160.633</td>
<td>457.750</td>
</tr>
<tr>
<td>4</td>
<td>6.412 %</td>
<td>158.099</td>
<td>407.856</td>
</tr>
</tbody>
</table>

Table 4.2: Robot State age using periodic transmissions, with network load
Time values are in milliseconds.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Packet loss</th>
<th>Ignoring transmission</th>
<th>Considering transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>0.000 %</td>
<td>150.646</td>
<td>256.612</td>
</tr>
<tr>
<td>2</td>
<td>0.151 %</td>
<td>154.699</td>
<td>299.998</td>
</tr>
<tr>
<td>3</td>
<td>0.000 %</td>
<td>159.591</td>
<td>456.883</td>
</tr>
<tr>
<td>4</td>
<td>0.000 %</td>
<td>154.915</td>
<td>297.578</td>
</tr>
</tbody>
</table>

Table 4.3: Robot State age using Adaptive-TDMA, without network load
Time values are in milliseconds.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Packet loss</th>
<th>Ignoring transmission</th>
<th>Considering transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>6.891 %</td>
<td>159.734</td>
<td>419.215</td>
</tr>
<tr>
<td>2</td>
<td>4.626 %</td>
<td>184.626</td>
<td>491.055</td>
</tr>
<tr>
<td>3</td>
<td>5.725 %</td>
<td>187.728</td>
<td>519.767</td>
</tr>
<tr>
<td>4</td>
<td>4.618 %</td>
<td>173.571</td>
<td>389.028</td>
</tr>
</tbody>
</table>

Table 4.4: Robot State age using Adaptive-TDMA, with network load
Time values are in milliseconds.
4. Evaluation

4.2 Evaluation of Synchronization Messages

Some synchronization messages carry information that will be important for the receivers. On the other hand, others carry questions, for which only the answers will be important for the initial sender. Synchronization messages should be evaluated by the time it takes for all the receivers to get the question, and how long it takes for the initial agent to receive all the answers.

The handler delay is the amount of time between the start of a question round and the reception of the question by the destination agent. Since this events happen on different computers, the accuracy of the values depends on clock synchronization.

The callback delay is the amount of time between the start of a question round and the reception of all the answers. This value does not depend on clock synchronization.

Four scenarios were considered: using our precious solution and using short rounds, both with and without network load. For each scenario, two experiments were made:

- **Single agent**: agent with static ID 0 sends a message to one other agent and receives the answer. The destination agent is chosen from the other team members in turn.

- **All other agents**: agent with static ID 0 sends a message to all other agents and receives all the answers.

The questions transmitted contain only a random number, which is repeated in the answers. This number is used to guarantee that the received answers are for the question that was asked. This shows that the implementation is functioning properly. This never failed.

The previous solution used by our team to transmit synchronization messages consists simply of transmitting these messages included in the robot state. Table 4.5 shows the result of experiments run using this method. Table 4.6 shows the results of the same experiments done with network load.

Synchronization messages transmitted this way were started only after the previous one finished, since our previous solution did not support simultaneous resolution of more than one question. To simulate the worst case scenario, the synchronization messages were started only in the moment after a robot state message was transmitted.

For these experiments, robot state (including the synchronization messages) was transmitted using the long rounds developed in this solution. This should help proving that the short rounds are an advantage even over Adaptive-TDMA.

In an ideal scenario, without any delays, the agents would transmit 10 times per second. The transmission from each agent would happen exactly 20 milliseconds after the transmission from the previous one. In the worst case of this scenario, a message would have to wait 100 milliseconds to be transmitted, but then be immediately received by all agents. Therefore, the handler delay would be 100 milliseconds. Then the agents would transmit the answer in their time slots. If only one agent is required to answer, the possible values for the callback delay are 120, 140, 160 and 180 milliseconds, averaging 150 milliseconds. If all the agents are required to answer, the callback delay
is the maximum of these values, 180 milliseconds. These values are the best possible for the tests when transmitting synchronization messages together with robot state.

The proposed solution uses short rounds, especially dedicated to transmit synchronization messages. Table 4.7 shows the results of experiments using short rounds. Synchronization messages were started at a constant rate of 10 per second, whether or nor the last one had completed, since short rounds are prepared to deal with simultaneous question rounds. Table 4.8 shows the results of the same experiments done with network load.

Without any delays, the questions are transmitted immediately, so the handler delay would be 0 milliseconds in this ideal scenario. The agents would then transmit the answers, separated by the short waiting time $T_w$ of 1 millisecond. If only one agent was required to answer, the callback delay would be 1 millisecond. If all other agents were required, it would be 4 milliseconds. These are the best possible values. Note that, if a packet is lost, the slot time $T_s$ of 5 milliseconds is used instead and the callback delay grows considerably.

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

### Table 4.5: Synchronization messages delay when transmitted with robot state, without network load

<table>
<thead>
<tr>
<th>Destination</th>
<th>Single agent</th>
<th>All other agents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Handler Delay</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>100.824</td>
<td>100.487</td>
</tr>
<tr>
<td>Average</td>
<td>122.504</td>
<td>116.488</td>
</tr>
<tr>
<td>Maximum</td>
<td>440.807</td>
<td>431.995</td>
</tr>
<tr>
<td><strong>Callback Delay</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>102.954</td>
<td>145.564</td>
</tr>
<tr>
<td>Average</td>
<td>186.578</td>
<td>214.030</td>
</tr>
<tr>
<td>Maximum</td>
<td>456.939</td>
<td>509.755</td>
</tr>
</tbody>
</table>

Time values are in milliseconds.

### Table 4.6: Synchronization messages delay when transmitted with robot state, with network load

<table>
<thead>
<tr>
<th>Destination</th>
<th>Single agent</th>
<th>All other agents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Handler Delay</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>102.292</td>
<td>101.934</td>
</tr>
<tr>
<td>Average</td>
<td>137.711</td>
<td>136.935</td>
</tr>
<tr>
<td>Maximum</td>
<td>444.598</td>
<td>948.631</td>
</tr>
<tr>
<td><strong>Callback Delay</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>118.547</td>
<td>177.353</td>
</tr>
<tr>
<td>Average</td>
<td>205.960</td>
<td>247.494</td>
</tr>
<tr>
<td>Maximum</td>
<td>694.694</td>
<td>1256.729</td>
</tr>
</tbody>
</table>

Time values are in milliseconds.

### Table 4.7: Synchronization messages delay when transmitted using short rounds, without network load

<table>
<thead>
<tr>
<th>Destination</th>
<th>Single agent</th>
<th>All other agents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Handler Delay</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>1.148</td>
<td>1.098</td>
</tr>
<tr>
<td>Average</td>
<td>4.809</td>
<td>2.407</td>
</tr>
<tr>
<td>Maximum</td>
<td>219.287</td>
<td>51.470</td>
</tr>
<tr>
<td><strong>Callback Delay</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>3.096</td>
<td>6.976</td>
</tr>
<tr>
<td>Average</td>
<td>9.605</td>
<td>12.044</td>
</tr>
<tr>
<td>Maximum</td>
<td>221.303</td>
<td>102.012</td>
</tr>
</tbody>
</table>

Time values are in milliseconds.

### Table 4.8: Synchronization messages delay when transmitted using short rounds, with network load

<table>
<thead>
<tr>
<th>Destination</th>
<th>Single agent</th>
<th>All other agents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Handler Delay</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>1.148</td>
<td>1.374</td>
</tr>
<tr>
<td>Average</td>
<td>16.655</td>
<td>22.227</td>
</tr>
<tr>
<td>Maximum</td>
<td>325.225</td>
<td>358.010</td>
</tr>
<tr>
<td><strong>Callback Delay</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>4.090</td>
<td>13.368</td>
</tr>
<tr>
<td>Average</td>
<td>30.207</td>
<td>69.924</td>
</tr>
<tr>
<td>Maximum</td>
<td>463.407</td>
<td>396.034</td>
</tr>
</tbody>
</table>

Time values are in milliseconds.
4.3 Evaluation of Interference

Robot state diffusion has been tested in section 4.1. However, in those tests, no short rounds were active. The interference that short rounds cause in long rounds must also be evaluated.

Tables 4.9 and 4.10 show the results of the same experiments done for tables 4.3 and 4.4 but with short rounds active. These short rounds are exactly the same as those used in section 4.2 with all other robots as destination.

The results show that network load clearly increases the interference that short rounds cause in long rounds. This was expected since packet loss causes question rounds to be resolved much slower. Consequently, since the long rounds drop the packet to be transmitted the first time they detect the short rounds are active, packet loss increases considerably in scenarios with short rounds active.

On the other hand, since the long rounds only drop the first packet, the average time values increase only slightly.
4. Evaluation

<table>
<thead>
<tr>
<th>Agent</th>
<th>Packet loss</th>
<th>Ignoring transmission</th>
<th>Considering transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Maximum</td>
</tr>
<tr>
<td>1</td>
<td>0.319 %</td>
<td>150.510</td>
<td>304.802</td>
</tr>
<tr>
<td>2</td>
<td>0.486 %</td>
<td>153.164</td>
<td>356.470</td>
</tr>
<tr>
<td>3</td>
<td>0.162 %</td>
<td>153.018</td>
<td>387.607</td>
</tr>
<tr>
<td>4</td>
<td>0.486 %</td>
<td>153.387</td>
<td>371.343</td>
</tr>
</tbody>
</table>

Table 4.9: Robot State age with low load of short rounds, without network load
To be compared with table 4.4. Time values are in milliseconds.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Packet loss</th>
<th>Ignoring transmission</th>
<th>Considering transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Maximum</td>
</tr>
<tr>
<td>1</td>
<td>11.389 %</td>
<td>166.786</td>
<td>434.588</td>
</tr>
<tr>
<td>2</td>
<td>7.110 %</td>
<td>188.439</td>
<td>536.186</td>
</tr>
<tr>
<td>3</td>
<td>9.375 %</td>
<td>192.171</td>
<td>532.957</td>
</tr>
<tr>
<td>4</td>
<td>10.000 %</td>
<td>186.540</td>
<td>618.945</td>
</tr>
</tbody>
</table>

Table 4.10: Robot State age with low load of short rounds, with network load
To be compared with table 4.4. Time values are in milliseconds.
4.4 Evaluation of Degradation

Synchronization messages are expected to be occasional, thus the solution is optimized for this case. However, the performance should degrade gracefully with the increase of synchronization messages.

The experiments for this solution done in sections 4.1 and 4.2 were run again. However, this time, overloading the protocol with synchronization messages.

This was accomplished by continuously starting as much question rounds as possible. This was done in all the agents, having all other agents required to answer. The protocol implementation currently limits the number of simultaneous question rounds started in one agent to 32. Still, this allowed for almost 1000 question rounds to be started and completed in each agent every second.

Tables 4.11 and 4.12 show the test results for robot state diffusion. As expected, the average age of information increased considerably.

Tables 4.13 and 4.14 show the test results for synchronization messages. The synchronization messages used to collect these results were the same as used in section 4.2, but marked to be distinguished from the messages used to overload the protocol. Again, as expected, the handler and callback delays are increased.

Although these tests stressed the protocol to its limits, it remained usable and dependable.
4. Evaluation

<table>
<thead>
<tr>
<th>Agent</th>
<th>Packet loss</th>
<th>Ignoring transmission</th>
<th>Considering transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Maximum</td>
</tr>
<tr>
<td>1</td>
<td>8.923 %</td>
<td>162.327</td>
<td>705.219</td>
</tr>
<tr>
<td>2</td>
<td>12.298 %</td>
<td>175.360</td>
<td>604.791</td>
</tr>
<tr>
<td>3</td>
<td>9.253 %</td>
<td>180.789</td>
<td>527.612</td>
</tr>
<tr>
<td>4</td>
<td>9.740 %</td>
<td>171.375</td>
<td>467.400</td>
</tr>
</tbody>
</table>

Table 4.11: Robot State age with heavy load of short rounds, without network load
To be compared with tables 4.3 and 4.9 Time values are in milliseconds.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Packet loss</th>
<th>Ignoring transmission</th>
<th>Considering transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Maximum</td>
</tr>
<tr>
<td>1</td>
<td>44.911 %</td>
<td>255.465</td>
<td>1229.041</td>
</tr>
<tr>
<td>2</td>
<td>36.716 %</td>
<td>251.182</td>
<td>979.296</td>
</tr>
<tr>
<td>3</td>
<td>42.193 %</td>
<td>268.822</td>
<td>872.779</td>
</tr>
<tr>
<td>4</td>
<td>43.633 %</td>
<td>274.903</td>
<td>1083.733</td>
</tr>
</tbody>
</table>

Table 4.12: Robot State age with heavy load of short rounds, with network load
To be compared with tables 4.4 and 4.10 Time values are in milliseconds.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Single agent</th>
<th>All other agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handler Delay</td>
<td>Minimum</td>
<td>1.987</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>18.902</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>76.962</td>
</tr>
<tr>
<td>Callback Delay</td>
<td>Minimum</td>
<td>9.183</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>30.782</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>133.411</td>
</tr>
</tbody>
</table>

Table 4.13: Synchronization Messages delay with heavy load of short rounds, without network load
To be compared with table 4.7 Time values are in milliseconds.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Single agent</th>
<th>All other agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handler Delay</td>
<td>Minimum</td>
<td>5.541</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>44.334</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>475.776</td>
</tr>
<tr>
<td>Callback Delay</td>
<td>Minimum</td>
<td>21.591</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>82.087</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>657.761</td>
</tr>
</tbody>
</table>

Table 4.14: Synchronization Messages delay with heavy load of short rounds, with network load
To be compared with table 4.8 Time values are in milliseconds.
Conclusions and Future Work

Contents

5.1 Conclusions ................................................................. 44
5.2 Future Work ............................................................... 45
5. Conclusions and Future Work

5.1 Conclusions

The proposed solution 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. Transmitting synchronization messages using some completely separate protocol would always cause unnecessary conflicts. For instance, using TCP over unicast channels to all team members would cause more probable collisions and repetition, since the transmission would have to be repeated for every team member. Using a central dispatcher requires an extra level of indirection and keeps the problems of collisions and repetitions, although in this case information is only repeated twice.

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 ROS. It can even be used outside the scope of RoboCup MSL, e.g., if a team of robots needs to be deployed in a situation where a public wireless network must be used, this solution would provide adequate communication capabilities.
5.2 Future Work

Some possibilities can be explored to enhance the proposed solution:

- At the moment, static IDs have to be given to the agents. This is an adequate solution for RoboCup MSL, where every robot has a number. However, in other scenarios, it might be interesting to deploy a team of robots without individual identification at this level. E.g., a group of robots carrying sensors could simply join the group and publish information. If the robots are localized, this information could be indexed by the position where it is captured and not by which specific robot captured it. Note that, in this scenario, synchronization messages only make sense if they target all agents. This could be used, for instance, to tell all robots close to a position to focus it in detail, or to terminate an experiment by calling all robots back.

- Time synchronization could be included in the protocol. Having this for granted, mechanisms based on time could be explored to coordinate the long rounds. The delay of packets could also be more accurately estimated. The agents could agree on a time to transmit, and even if several packets would be lost, the agents would not lose synchronization.
5. Conclusions and Future Work
Bibliography


Bibliography


