1-1-2013

Reducing Communication Delay Variability for a Group of Robots

Goncalo Martins
University of Denver

Follow this and additional works at: https://digitalcommons.du.edu/etd

Part of the Artificial Intelligence and Robotics Commons, and the Robotics Commons

Recommended Citation
Martins, Goncalo, "Reducing Communication Delay Variability for a Group of Robots" (2013). Electronic Theses and Dissertations. 400.
https://digitalcommons.du.edu/etd/400

This Dissertation is brought to you for free and open access by the Graduate Studies at Digital Commons @ DU. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ DU. For more information, please contact jennifer.cox@du.edu,dig-commons@du.edu.
Reducing Communication Delay Variability for a Group of Robots

A Dissertation
Presented to
the Faculty of the Daniel Felix Ritchie
School of Engineering and Computer Science
University of Denver

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Goncalo Martins
November 2013
Advisors: Kimon P. Valavanis Ph.D. and Matthew J. Rutherford Ph.D.
Abstract

A novel architecture is presented for reducing communication delay variability for a group of robots. This architecture relies on using three components: a microprocessor architecture that allows deterministic real-time tasks; an event-based communication protocol in which nodes transmit in a TDMA fashion, without the need of global clock synchronization techniques; and a novel communication scheme that enables deterministic communications by allowing senders to transmit without regard for the state of the medium or coordination with other senders, and receivers can tease apart messages sent simultaneously with a high probability of success. This approach compared to others, allows simultaneous communications without regard for the state of the transmission medium, it allows deterministic communications, and it enables ordered communications that can be applied in a team of robots. Simulations and experimental results are also included.
Acknowledgments

It is always difficult to write a gratitude text when the number of persons involved is too big. It does not mean that the others that are not referred do not deserve my attention and recognition as well.

To my mom and dad, for the sacrifices supported that made possible my graduation and for all the comfort and affection that they have always revealed.

My thanks to my advisors Dr. Kimon Valavanis, Dr. Rita Ribeiro and Dr. Matthew Rutherford, for the orientation and long discussions not only about the contents of this dissertation but also for the resources and human opportunities provided.

For all my colleagues and friends, companions in this path, by their camaraderie and solidarity shown, not only by the encouragement that they always provided but also by the long conceptual and philosophical discussions taken through along these years. The difficult moments will be better overcome together.

This research was partially financed by a PhD grant (SFRH/BD/60571/2009) given by the Portuguese Foundation FCT - Fundação para a Ciência e a Tecnologia, Portugal.
# Table of Contents

1 Introduction  ............................................. 1  
   1.1 Problem Statement ..................................... 2  
   1.2 Method of Approach ................................... 5  
   1.3 Research Contributions ............................... 6  
   1.4 Thesis Outline ....................................... 7  

2 Background Information ................................. 9  
   2.1 Mobile Robots ......................................... 9  
      2.1.1 Multi-Robot System ............................. 11  
      2.1.2 Multi-Robot Architecture ..................... 11  
   2.2 Formation Control Methods .......................... 13  
   2.3 Ad Hoc Networks ...................................... 14  
      2.3.1 MANET ........................................ 14  
      2.3.2 Wireless Sensor Networks .................... 14  
      2.3.3 MAC Schemes ................................ 15  

3 Literature Review ..................................... 18  
   3.1 Existing Microcontrollers ............................ 19  
   3.2 Network Control Systems ............................. 19  
   3.3 MANETs MAC Protocols ............................... 24  
      3.3.1 Contention Based MAC Protocols ............... 25  
      3.3.2 Allocation Based MAC Protocols .............. 26  
      3.3.3 Hybrid MAC Protocols ........................ 26  
   3.4 WSN MAC Protocols .................................. 27  
   3.5 Formation Network Techniques ....................... 28  
   3.6 Time Synchronization Techniques .................... 30  
      3.6.1 Ad Hoc Synchronization Protocols ............ 31  
   3.7 Heuristics for Communication Delays ............... 32  

4 Proposed Architecture ................................. 34
# 5 Technology Selection

5.1 Introduction ........................................... 41
5.2 Brief History ........................................... 42
5.3 XMOS .................................................... 44
5.4 Existing Technologies ................................. 45
5.5 Benchmark .............................................. 46
   5.5.1 Introduction ...................................... 46
   5.5.2 Conventional Benchmarks ......................... 48
   5.5.3 Proposed Methodology ............................. 50
   5.5.4 Platforms Under Test .............................. 55
   5.5.5 Evaluation ......................................... 57

# 6 Robots Scheduling

6.1 Introduction ........................................... 61
6.2 RT-MAC .................................................. 62
   6.2.1 Requirements ...................................... 64
   6.2.2 Frame Description ................................. 65
   6.2.3 Scheduling .......................................... 67
   6.2.4 Behavior State Machines ......................... 68
   6.2.5 Hybrid Group Control .............................. 72
   6.2.6 Protocol Analysis .................................. 72
6.3 Time-Line Examples .................................... 76
   6.3.1 Create Group ...................................... 79
   6.3.2 Lost Leader / Assign New Leader ................. 79
   6.3.3 Join Group ......................................... 80
   6.3.4 Leave Group ....................................... 81
6.4 Simulation .............................................. 82
   6.4.1 Simulation Software ............................... 82
   6.4.2 Setup and Performance Metrics ................... 83
   6.4.3 Results ............................................ 84
   6.4.4 Discussion and Analysis ......................... 86
6.5 Hardware Implementation ............................. 87
9.4  Time-Line Examples ................................................. 132
  9.4.1  Create Group .................................................. 133
  9.4.2  Lost Leader / Assign New Leader .......................... 134
  9.4.3  Join Group ...................................................... 134
  9.4.4  Leave Group .................................................... 135
9.5  Hardware Implementation .......................................... 137

10  Conclusions and Future Work ................................. 138
  10.1  Conclusion ...................................................... 138
  10.2  Future Work .................................................... 139
  10.3  Publications .................................................... 140
### List of Figures

1.1 Shape Formation Example ........................................... 3  
1.2 Position Error Example ............................................ 5  
2.1 Unmanned Systems Examples - (a)UGV (b)UAV (c)UUV .......... 10  
2.2 Digital data transmitted using FDMA [37] ...................... 16  
2.3 Digital data transmitted using TDMA [37] ...................... 17  
3.1 Existing microcontrollers for multi-robot systems. (a)UAV V3 (b)Gluonpilot (c)Paparazzi (d)Gumstix (e)Ardupilot (f)Ardupilot Mega ...... 20  
3.2 Comparison of scaled step responses with full-state feedback [21] ... 22  
3.3 Step response for different communication delays [41] .......... 23  
3.4 The system state response with 1.05s time delay [42] .......... 23  
3.5 Comparison of Different Event Arrival Models [43] .......... 24  
3.6 MAC Protocols for MANETs ........................................ 27  
3.7 MAC Protocols for WSN [52] .................................... 28  
3.8 WSN Synchronization Protocols [52] .......................... 32  
4.1 Synchronization delay between a pair of nodes [52] .......... 35  
4.2 Example of multiple tasks that can be running simultaneously in a microcontroller ............................................. 36  
4.3 Architecture Requirements ......................................... 40  
5.1 XMOS Architecture (from www.xmos.com/) .................... 44  
5.2 a) XBS Framework b) Proposed Framework ..................... 50  
5.3 Measurement Response Time Example .......................... 53  
5.4 Worst Case Performance Vs Number of Tasks (Normalized Results) 57  
5.5 Best Case Performance Vs Number of Tasks (Normalized Results) 58  
5.6 Jitter Performance Vs Number of Tasks (Normalized Results) 58  
6.1 RT-MAC place in Ad Hoc Networks Protocols ................ 63  
6.2 The RT-MAC Frame ................................................ 65  
6.3 RT-MAC Frame with Packet Repetition (PR) ................... 66
6.4 Frame with Static and Dynamic Slots ............................................. 68
6.5 Behavior State Machine ................................................................. 69
6.6 Submachine states: LearnScheduler and CreateNew ......................... 70
6.7 Submachine states: JoinGroup and LeaveGroup ............................... 70
6.8 Submachine state: GroupActive ....................................................... 71
6.9 N vs DTS (fixed FL, GP, PR and MTR) ........................................... 75
6.10 N vs DTS for GP = DTT (fixed FL, PR and MTR) ............................ 76
6.11 ND\textsubscript{DS} vs DTS\textsubscript{DS} for (fixed DS\textsubscript{Time}, GP and MTR) .................................................. 77
6.12 RT-MAC Scheduler - Basic Example .............................................. 78
6.13 RT-MAC Scheduler - Create a Group .............................................. 79
6.14 RT-MAC Scheduler - Lost Leader / Assign New Leader ...................... 79
6.15 RT-MAC Scheduler - One Robot Join Example .................................. 80
6.16 RT-MAC Scheduler - Three Robots Join Example ............................ 81
6.17 RT-MAC Scheduler - Robot Join Failed Example ............................. 81
6.18 RT-MAC Scheduler - One Robot Leave Example ................................ 82
6.19 RSS Results with unordered transmissions .................................... 84
6.20 LT Results with unordered transmissions ..................................... 85
6.21 LT Results with ordered transmissions ......................................... 86
6.22 RT-MAC Node with XMOS board and WiFly module ......................... 87
6.23 RT-MAC place in the OSI model with WiFi as under layer ................ 88
6.24 RT-MAC Nodes ............................................................................ 88
7.1 OR-channel result ........................................................................... 95
7.2 Acquisition with No Repetition ....................................................... 98
7.3 Acquisition with Repetition ............................................................ 99
7.4 iterDecode State Machine ............................................................... 101
7.5 Hallucination Rate vs #Nodes (Simulation) ...................................... 104
7.6 #Repetitions vs #Nodes (Simulation) .............................................. 104
7.7 BBC Tasks ..................................................................................... 106
7.8 Wired Physical Setup ...................................................................... 108
7.9 Wireless Physical Setup .................................................................. 112
8.1 BBC Tasks + Behavior Tasks ........................................................... 116
8.2 BBCDecoder Timing vs # Nodes Transmitting Simultaneously (XMOS) 118
8.3 Time to Transmit 1 Message vs # Nodes Transmitting Simultaneously (XMOS) ................................................................. 119
8.4 Time to Transmit 1 Message vs # Nodes Transmitting Simultaneously (Different Platforms) .......................... 120
8.5 Physical Setup ........................................... 122
8.6 BBC Node with XMOS board, AM emitter and receiver ............... 123
8.7 Difference Between Master and Slaves Packet Reception Time vs PDF (Density) - [Test 1] Beacon Time = 2s (Not Sync) .................. 125
8.8 Difference Between Master and Slaves Packet Reception Time vs CDF (Cumulative Density) - [Test 1] Beacon Time = 2s (Not Sync) .... 125
8.9 Difference Between Master and Slaves Packet Reception Time vs PDF (Density) - [Test 2] Beacon Time = 2s (Not Sync) ............... 126
8.10 Difference Between Master and Slaves Packet Reception Time vs CDF (Cumulative Density) - [Test 2] Beacon Time = 2s (Not Sync) . 126
8.11 Difference Between Master and Slaves Packet Reception Time vs PDF (Density) - [Test 3] Beacon Time = 2s (Sync) ....................... 127
8.12 Difference Between Master and Slaves Packet Reception Time vs CDF (Cumulative Density) - [Test 3] Beacon Time = 2s (Sync) .... 127
9.1 RT-MAC place in the OSI model with BBC as under layer ............ 130
9.2 BBC with RT-MAC Scheduler - Basic Example .......................... 132
9.3 BBC with RT-MAC Scheduler - Create a Group ........................ 133
9.4 BBC with RT-MAC Scheduler - Lost Leader / Assign New Leader .. 134
9.5 BBC with RT-MAC Scheduler - One Robot Join Example ............... 134
9.6 BBC with RT-MAC Scheduler - Three Robots Join Example .......... 135
9.7 BBC with RT-MAC Scheduler - Robot Join Failed Example ........... 136
9.8 BBC with RT-MAC Scheduler - One Robot Leave Example ............ 136
9.9 BBC with RT-MAC - Physical Setup ................................. 137
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Microcontrollers features</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Formation Network Techniques</td>
<td>30</td>
</tr>
<tr>
<td>7.1</td>
<td>Simulation tests description</td>
<td>103</td>
</tr>
<tr>
<td>7.2</td>
<td>Wired tests description</td>
<td>109</td>
</tr>
<tr>
<td>7.3</td>
<td>Simulation Vs Wired Results</td>
<td>110</td>
</tr>
<tr>
<td>7.4</td>
<td>Wireless tests description</td>
<td>111</td>
</tr>
<tr>
<td>7.5</td>
<td>Wireless Results</td>
<td>113</td>
</tr>
<tr>
<td>8.1</td>
<td>BBCDecoder decoding time (XMOS)</td>
<td>117</td>
</tr>
<tr>
<td>8.2</td>
<td>BBCDecoder decoding time (2nd (2GHz) and 6th (2.7GHz) Generation)</td>
<td>120</td>
</tr>
<tr>
<td>8.3</td>
<td>Tests Description</td>
<td>124</td>
</tr>
<tr>
<td>8.4</td>
<td>Tests Results</td>
<td>124</td>
</tr>
</tbody>
</table>


Acronyms

ACS  Adaptive Clock Synchronization

ADC  Analog to Digital Converter

ASIC  Application-Specific Integrated Circuit

CDF  Cumulative Distribution Function

CDMA  Code Division Multiple Access

CSMA  Carrier Sense Multiple Access

CTS  Clear-To-Send

DSTime  Dynamic Slot Time

DTT  Data Transmission Time

DTS  Data Transmission Size

DU2SRI  University of Denver Unmanned Systems Research Institute

FDMA  Frequency Division Multiple Access

FHMA  Frequency-Hopping Multiple Access

FL  Frame Length

GP  Guard Period

GPS  Global Positioning System

IMU  Inertial Measurement Unit
LTS  Lightweight Tree-Based Synchronization

MAC  Medium Access Control

MANET  Mobile Ad Hoc Network

MTR  Maximum Transmission Rate

NCS  Network Control Systems

NS2  Network Simulator 2

NTP  Network Time Protocol

OSI  Open Systems Interconnection

PR  Packet Repetition

PDF  Probability Density Function

PWM  Pulse-width Modulation

RBS  Reference-Broadcast Synchronization

RDP  Rate-Based Diffusion Protocol

RFID  Radio-frequency identification

RTOS  Real-Time Operating Systems

RTS  Request-To-Send

SDR  Software Defined Radios

SSTime  Static Slot Time
**TDMA**  Time Division Multiple Access

**TDP**  Time Diffusion Synchronization Protocol

**TPSN**  Timing-Sync Protocol for Sensor Networks

**UART**  Universal Asynchronous Receiver/Transmitter

**UAV**  Unmanned Aerial Vehicles

**UGV**  Unmanned Ground Vehicles

**UML**  Unified Modeling Language

**US**  Unmanned System

**USRP**  Universal Software Radio Peripheral

**UTC**  Coordinated Universal Time

**UUV**  Unmanned Underwater Vehicles

**WSN**  Wireless Sensor Networks
Chapter 1

Introduction

In the fields of mobile robots and unmanned systems, it is common for control systems and network protocols to be studied independently of each other. From the control systems perspective, a continuously available communications mechanism is assumed, ignoring the fact that the robots are mobile, and their motion will affect the communications performance. From the network protocol perspective, mobile ad hoc networks are often studied, but there is no couple between the communications and the control systems.

A team of robots or autonomous vehicles is normally modeled as a mobile ad hoc network where communication between the nodes and formation control are related to each other. These systems have sensors, actuators and computing elements that are related and interconnected. Moreover, the integration of ad hoc networks and control systems is a difficult task, and the study and design of a common architecture where both areas can be simultaneously designed and analyzed are a challenge.

The area of research that deals with the issues of combining sensors, actuators and computing elements that are connected by means of a network or other shared
medium is known as Network Control Systems (NCS). It addresses fundamental questions in communication, information processing, and control dealing with the relationship between operations of the network and the quality of the overall system’s operation. Mobile robots share the same fundamental questions.

A major concern in a group of mobile robots is the packet losses or delays present during communication. These issues can degrade the stability of control systems that for example affect the formation performance of the group.

This dissertation’s goal is to develop a deterministic controller architecture that reduces communication delay variability, which will allow a group of robots to simultaneously achieve reliable shape formations and communications.

1.1 Problem Statement

The formation control problem is defined as finding a control algorithm ensuring that multiple autonomous vehicles can uphold a specific formation or specific set of formations while traversing a trajectory and avoiding collisions simultaneously [1].

Figure 1.1 illustrates an example of a robot formation. It consists of 3 robots in a triangular formation in which the leader (red dot) must follow a predefined path. The remaining robots (blue dots) move with the leader and maintain the triangular formation. Each member of the formation updates its position based on the position of other members.

The control algorithms to ensure the shape formation can be implemented using several techniques such as: behavior-based control [2, 3, 4, 5, 6, 7, 8, 9, 10], graph theoretic [11, 12, 13, 14, 15], or virtual structures [16, 17, 18, 19]. Independent of the technique used, the main goal of each member is to follow a path, which is
Figure 1.1: Shape Formation Example
connected/related to the shape formation. Each member of the group formation needs to know the position of the other members in order to update its respective path. This can be achieved using global communication channels or local sensing (e.g., a vision system). This thesis focuses on the techniques that use communication channels for sharing location state within the group.

The control algorithms mentioned above need to include a communication channel model in their design to accommodate the failures present during communication. The communication channel model “helps” the control algorithms to follow the respective reference command or meet the requirements of the respective cost function. The more accurate the communication model is, the more accurate the controller will perform.

The control algorithms are also subject to error variation. Figure 1.2 shows a position error example. The blue area is what the control algorithms try to optimize or follow. The position error starts to increase (red area) as soon as the packets that contain location information of other members are not received. Once the packets are received, the controller can continue following the reference point (blue area again).

The control algorithms that assume perfect communication channels, generally wireless, are more vulnerable to delays and packet losses that may degrade the stability of the control system [20, 21]. In decentralized control one has a “local” view, which is ε (error margin from Figure 1.2) different from the “actual” state. To deal with these network delays, prediction techniques (e.g., Smith Predictor [22]) are normally added to the controller. However, the network delays are still present and the prediction techniques can only predict up to a certain number of received packets that will affect the desired delay margin boundaries.
Figure 1.2: Position Error Example

The direct consequence of large error margins ($\varepsilon$) on robot teams shape formation is that time has moved on, so the robots that are following are out of position. Once they get updated state information, there is still a finite amount of time before they are back in the correct position.

In a wireless environment, delays and packet losses are common and they may degrade the stability of control systems. These delays can affect the formation performance of a group of robots. Therefore, we study and analyze techniques or solutions that can minimize these network delays in order to achieve reliable formations.

1.2 Method of Approach

The main contribution is the proposed deterministic controller architecture that helps minimize network delays. The central objective is the overall reliability of group formation.
The proposed solution is based on the integration of a microprocessor architecture that enables deterministic processing times; implementation and optimization of an event-based communication protocol; and the development of a novel communication scheme to allow deterministic communication between a group of robots.

1.3 Research Contributions

The advantage and novelty of the proposed approach compared to similar ones is the integration of the control system with the network protocol. This is achieved by combining a microprocessor architecture with a communication protocol and a communication scheme that allows reliable communications among a group of robots.

**Microprocessor Architecture**

Mobile robots normally use microprocessors or embedded systems to handle the integration between sensors, actuators, computing elements and communication modules. Moreover, these embedded systems handle multiple tasks running simultaneously. This can be a time processing onerous task and it can introduce unpredictability. In this study, the use of a technology to overcome the lack of determinism is proposed.

**Robot Communication Protocol**

Network protocols for mobile robots rely on scheduling techniques (e.g., Time Division Multiple Access (TDMA)) to achieve reliable communications. Several Medium Access Control (MAC) protocols have been developed for groups of mobile robots [23, 24, 25, 26, 27]. The majority of these protocols are time-based, i.e., they rely on global clock synchronization to coordinate the data transmission between nodes. These clock synchronization techniques introduce delays during com-
munication. In this study, an event-based communication protocol is proposed, in which the nodes transmit in a TDMA fashion (i.e., only during their designated time slot), without the need of global clock synchronization techniques.

**MAC Communication Scheme**

A major concern in network scheduling techniques is the management of the shared transmission medium [28, 29]. Wireless communications suffer from self-jamming as a consequence of multiple nodes attempting to transmit at the same time.

In this thesis, a novel communication scheme that enables deterministic communications is proposed by allowing senders to transmit without regard for the state of the medium or coordination with other senders, and receivers can tease apart messages sent simultaneously with a high probability of success.

Moreover the proposed robot communication protocol can take advantage of this new communication scheme to improve some features, such as the transmission of simultaneous messages or the simplification of the join state.

### 1.4 Thesis Outline

This dissertation consists of 10 chapters. Chapter 1 presents the motivation, problem statement, method of approach and research contributions. Chapter 2 provides the necessary background information about mobile robot systems, existing microprocessor architectures, formation control strategies and formation network techniques. Chapter 3 provides the literature review. The most relevant microprocessors architectures and communication protocols are discussed and a comparative study is given. Chapter 4 presents the problem formulation and details about the
proposed deterministic architecture in order to enable reliable formations. In Chapter 5, the microprocessor architecture is discussed. Chapter 6, presents the description and simulation results of the proposed communication protocol. The description and simulation results of the novel communication scheme are presented in Chapter 7. Chapter 8 presents the implementation of the communication scheme as a MAC layer. Chapter 9 demonstrates the integration of the microprocessor architecture with the communication protocol and scheme. Finally, Chapter 10 presents the conclusions of this research and discusses future work.
Chapter 2

Background Information

This Chapter provides the necessary background knowledge for this research. A brief description of mobile robots is given in Section 2.1. Formation control methods and ad hoc networks are described in Section 2.2 and Section 2.3, respectively.

2.1 Mobile Robots

A mobile robot is considered to be unmanned when there is no presence of a human to control it and it is capable of operating under remote control or autonomously. They can also be considered Unmanned System (US). They come in different shapes and sizes and can be classified into 3 main groups: Unmanned Ground Vehicles (UGV), Unmanned Aerial Vehicles (UAV) and Unmanned Underwater Vehicles (UUV). Figure 2.1 shows an example of a US for each group mentioned above.

A mobile robot or an US is composed by mechanical components (i.e., motors, frame structure) and integrated peripherals (i.e., Global Positioning System (GPS),
Figure 2.1: Unmanned Systems Examples - (a) UGV (b) UAV (c) UUV
Inertial Measurement Unit (IMU), Wifi, microcontroller) which enable navigation and communications.

2.1.1 Multi-Robot System

A single mobile robot might not be sufficient to accomplish certain tasks (e.g., planet exploration, pushing objects) by itself. For these tasks, several mobile robots can be utilized to accomplish a task that would otherwise be difficult or impossible for a single robot. A group of mobile robots is called a multi-robot system.

A multi-robot team is composed by homogeneous or heterogeneous systems ranging from slightly different sensors to entirely distinct hardware and/or software platforms. It can be classified into 2 groups:

- Cooperative robot teams: each robot generally has different capabilities and control algorithms which when combined can be used to complete a task;
- Swarm of robots: each robot generally has identical function and capabilities with the goal being the overall group behavior.

2.1.2 Multi-Robot Architecture

The group architecture of a multi-robot team provides the framework upon which missions are implemented, and determines the system functionality and boundaries. The key architectural aspects of a group team includes: type of control, team composition, sensing and signaling, communication, physical interaction, power and size [30].

Type of Control

There are two main types of control:
• Centralized control in which individual robots receive commands from a central controller;

• Decentralized control in which local control laws operating in individuals robots produce a desired global, emergent behavior. Decentralized control methods follow two forms:
  – Distributed control: all robots are equal with respect to control;
  – Hierarchical control: control is locally centralized.

Decentralized control methods are advantageous over centralized ones in that they are more fault tolerant, scalable and reliable [30].

**Team Composition**

• Homogeneous: if the capabilities of the individual robots are identical. All robots have equal capabilities and priorities;

• Heterogeneous: any difference in software or hardware can make a robot different from another. It is necessary to prioritize a robot’s tasks based on its capabilities.

**Sensing and Signaling**

The main emphasis in multi-robot systems is the interaction among the robots as well as the interaction of the robots with their environment, resulting in extra constraints for the robots to be used. In particular:

• The interface among the sensing systems of the robots and the effect of environmental factors on them should be minimal;
• The robots should be able to distinguish between other kind of robots;

• The robots should be able to leave “marks” in the environment and be able to subsequently sense them.

Communication

Unlike stand-alone robotic systems, communication by plugging cables into the robots is no longer a feasible option. Therefore mobile robots have to support means to communicate (i.e., wireless communications, visual communication).

Power

The robots should have a long battery life. In most studies, the swarm may need to operate for a period that is long enough for the collective behavior to emerge, and the goal to be reached.

2.2 Formation Control Methods

The shape formation of a multi-robot team is organization of the robots into a particular shape. This shape or formation can be task and environment dependent. Current applications where pattern formation is implemented include convoy support [2], chemical source localization [31], and UAV formations control [32, 33, 34].

In formation control for a group of robots, different control methods can be adopted depending on the specific scenarios and/or missions. Various approaches to formation control are behavior-based control [4, 6, 35], graph theoretic [35, 11, 36], and virtual structures [18, 16].
2.3 Ad Hoc Networks

The term “ad hoc” is a Latin expression that means for this or more specifically, for this purpose only. This term is used in wireless networks consisting of communication entities that belong to a network only during a communication session and are within a radio range with other entities. These connections among entities are usually temporary since nodes may be added or removed either logically or physically. In general, ad hoc networks operate in a standalone way, but may be connected to another network such as the Internet.

There are two kinds of ad hoc networks: Mobile Ad Hoc Network (MANET) and Wireless Sensor Networks (WSN) [37].

2.3.1 MANET

A MANET consists of mobile hosts that communicate with each other using wireless links. A MANET is a self-configuring network that can have an arbitrary topology over the time and can change quickly and unpredictably since there may exist a large number of independent ad hoc connections.

MANETs play an important role in new distributed applications such as distributed collaborative computing, distributed sensing applications, next generation wireless systems, and response to incidents without a communication infrastructure [37].

2.3.2 Wireless Sensor Networks

WSN are networks in which nodes are low-cost sensors that can communicate with each other in a wireless manner, have limited computing capability, memory
and operate with limited battery power [38]. The main goal of such networks is to perform distributed sensing tasks, particularly for applications like environmental monitoring, smart and medical systems, that is, in this network, all nodes are often involved in the same sensing task [37].

Different from nodes in customary ad hoc network, sensors are generally stationary after deployment. Despite the fact that nodes are static, these networks still have a dynamic network topology. Some characteristics of wireless sensor networks are presented next:

• During periods of low activity the nodes go to sleep to conserve energy;

• The nodes go out of service when the energy of the battery runs out or when a destructive event takes place;

• Sensors have limited resources, such as limited computing capability, memory and energy supplies; they must balance these restricted resources to increase the lifetime of the network;

• The nodes will be battery-powered and it is often very difficult to change or recharge batteries.

For these reasons, in WSNs, there is interest in prolonging the lifetime of the network and, thus, energy conservation is one of the most important aspects to be considered in the design of such networks.

2.3.3 MAC Schemes

Wireless networks include a vast range of networks such as cellular networks, mobile ad hoc networks, sensor networks, satellite systems, and wireless local area

15
networks. A common feature of all these wireless networks is a broadcast channel as the basis of their communication.

A broadcast channel is a single communication channel that is shared by all other nodes. This is an advantage if the packet is addressed to all destinations. But if there are multiple concurrent senders how can they share the same channel while avoiding packet collisions? The answer relies on the use of Medium Access Control (MAC) protocols to determine which node transmits next on the broadcast channel.

Frequency, time, and spread spectrum multiple access are the three major access schemes used to share the available bandwidth in a wireless communication system [37].

**Frequency Division Multiple Access (FDMA)**

Frequency Division Multiple Access (FDMA) assigns individual channels to individual nodes. Figure 2.2 shows that in FDMA, each node is allocated a unique frequency band or channel. FDMA is a continuous transmission scheme and it is suitable for both analog and digital systems.

![Figure 2.2: Digital data transmitted using FDMA [37]](image)

**Time Division Multiple Access (TDMA)**


In TDMA systems, the radio spectrum is divided into non-overlapping time slots. Figure 2.3 shows TDMA, where only one user is allowed to transmit in each slot. Because the frame cyclically repeats, a channel for a user has a particular time slot that recurs in every frame. As a result, the transmission for any user is not continuous.

![Figure 2.3: Digital data transmitted using TDMA](image)

With TDMA, it is possible to dynamically allocate different time slots per frame to different users. Thus, bandwidth can be supplied on demand to different users.

**Spread Spectrum Multiple Access**

Spread spectrum multiple access communication uses signals that have a bandwidth that is several orders of magnitude greater than the minimum required bandwidth. Such wide band systems are efficient when many users share the bandwidth without interfering with one another [37]. There are two forms of spread spectrum multiple access schemes: Frequency-Hopping Multiple Access (FHMA) spreads the signal over time and narrow band frequency channels, and direct sequence multiple access (also called Code Division Multiple Access (CDMA)), where the signal energy is spread over the wide band channel.
Chapter 3

Literature Review

This Chapter starts with the description of existing microcontroller architectures commonly used in multi-robot systems. Section 3.2 describes the communication delays present in Network Control Systems. Section 3.3 and Section 3.4, present different MAC techniques applied to MANETs and WSN protocols, respectively. A comparative study on existing formation network techniques is presented in Section 3.5. Multi-robot systems focusing on formation network techniques are included in this comparison. The formation network techniques are analyzed according to their MAC protocol technique. In Section 3.6, it is described time synchronization issues in ad hoc networks that are time-based and rely on a global clock and it is presented existing clock synchronization protocols. Finally, Section 3.7 discusses heuristic methods for communication delays.
### 3.1 Existing Microcontrollers

Multi-robot systems, commonly have multiple tasks running and interacting simultaneously between each other and significant I/O interaction that must operate in consort with these tasks.

To handle all these tasks and I/O interactions, multi-robot systems are equipped with embedded systems (i.e., microcontrollers or microprocessors) that are responsible to manage all these resources. There are different types of microcontrollers, and the most commonly used architectures in multi-robot systems are the following: PIC from Microchip, ARM architecture, Atmel / Atmega (Arduino), PC104, x86 architecture and FPGAs [39].

Figure 3.1 shows some existing microcontrollers and Table 3.1 presents information about different platforms used in multi-robot systems.

Some of these embedded systems require software to respond to many inputs and events within tight time constraints (in “real-time”). Real-Time Operating Systems (RTOS) are designed to manage the execution of this software in embedded systems.

With such a variety of embedded systems, which ones are more suitable to use for multi-robot systems? In Chapter 4, it is described what are the desired requirements of such system in order to be used for multi-robot systems and in Chapter 5, it is presented a comparative study between a couple of architectures in order to highlight the importance of relevant features.

### 3.2 Network Control Systems

As mentioned in Chapter 1 (Section 1.1), this thesis focuses on the techniques that use communication channels for sharing location state within the group. The
Figure 3.1: Existing microcontrollers for multi-robot systems. (a) UAV V3 (b) Gluonpilot (c) Paparazzi (d) Gumstix (e) Ardupilot (f) Ardupilot Mega
<table>
<thead>
<tr>
<th>Name</th>
<th>Microcontroller</th>
<th>Systems</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAV V3</td>
<td>dsPIC30F4011</td>
<td>UAVs</td>
<td>3-axis accelerometer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 dual axis gyros</td>
</tr>
<tr>
<td>Gluonpilot</td>
<td>dsPIC</td>
<td>UAVs</td>
<td>Supports up to 6 RC Channels</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pressure Sensor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No magnetometer</td>
</tr>
<tr>
<td>NAVGo V3</td>
<td>LPC2148</td>
<td>UAVs</td>
<td>Barometer</td>
</tr>
<tr>
<td>(Paparazzi)</td>
<td></td>
<td></td>
<td>MEMs IMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Magnetometer</td>
</tr>
<tr>
<td>Krooz</td>
<td>STM32F405</td>
<td>UAVs</td>
<td>Barometer</td>
</tr>
<tr>
<td>(Paparazzi)</td>
<td></td>
<td></td>
<td>MEMs IMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Magnetometer</td>
</tr>
<tr>
<td>Draganflyer</td>
<td>TI OMAP</td>
<td>UAVs</td>
<td>Servo Controller for up to 6 servos</td>
</tr>
<tr>
<td>(Gumstix)</td>
<td>Marvell Scale</td>
<td></td>
<td>3x UART on 4-pin PicoBlade</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1x I2C on 4-pin PicoBlade</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPIO connectors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3x USB on 4-pin PicoBlade</td>
</tr>
<tr>
<td>Pixhawk</td>
<td>Intel Core 2 DUO</td>
<td>UAVs</td>
<td>Servo Controller for up to 6 servos</td>
</tr>
<tr>
<td></td>
<td>Gumstix Overo COM</td>
<td></td>
<td>3x UART on 4-pin PicoBlade</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1x I2C on 4-pin PicoBlade</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPIO connectors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3x USB on 4-pin PicoBlade</td>
</tr>
<tr>
<td>Ardupilot</td>
<td>Atmega</td>
<td>UAVs</td>
<td>No sensors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UGVs</td>
<td>6-pin GPS connector</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 spare analog inputs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hardware-driven servo control</td>
</tr>
<tr>
<td>Ardupilot</td>
<td>Atmega</td>
<td>UAVs</td>
<td>3 axis gyros</td>
</tr>
<tr>
<td>Mega</td>
<td></td>
<td>UGVs</td>
<td>3 axis accel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Barometer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Data logging</td>
</tr>
</tbody>
</table>

Table 3.1: Microcontrollers features

control algorithms to ensure the shape formation (e.g., behavior-based control, graph theoretic or virtual structures) need to include a communication model in their design to accommodate the failures present during communication.
The insertion of the communication network in feedback control loops makes the analysis and design of a NCS more complex, and introduces some issues that degrade the control system’s performance [40]. The authors in [21, 41] present some examples of the effect of communication delays on control systems (Figure 3.2 and 3.3, respectively).

![Figure 3.2](image.png)

Figure 3.2: Comparison of scaled step responses with full-state feedback [21]

Thus, it is important to pay attention to the network communication protocols and the congestion control policy suitable for NCS.

There are 3 possible approaches the control algorithms described above deal with communication delays:

**Assume Perfect Communications**

The control algorithms assume perfect communications. This is the simplest and easiest solution. However, in real systems, there is no guarantee for non-delayed information sent from sensors to controllers or control signals from controllers to actuators [40]. The probability of failure is higher.
Figure 3.3: Step response for different communication delays [41]

Assume a Constant Communication Delay

The authors in [42] present the system state response with a constant time delay (Figure 3.4). It is a conservative method and if the communication environment changes it might not give the optimal result.

Figure 3.4: The system state response with 1.05s time delay [42]
Communication Environment Model

Trying to model the communication environment is the most challenging one. The author in [43] presents a comparison of different packets arrival models (Figure 3.5). It will give the best results of the 3 approaches but if the communication environment changes the model might not be applicable.

![Figure 3.5: Comparison of Different Event Arrival Models [43]](image)

To accommodate these environment changes, prediction techniques (e.g., Kalman Filter or Smith Predictor) can also be included in the model [22].

The more accurate is the communication model the more accurate the controller will perform.

3.3 MANETs MAC Protocols

The inclusion of communication models in the feedback loop controller might be complex or it might not be enough to overcome the problem of latency and packets
loss. One of the main reasons is the access to the same transmission medium that influences the communications and the integration of that access into the communication model is a difficult task. A team of mobile robots is normally modeled as a mobile ad hoc network. In ad hoc networks MAC protocols are used to minimize communication delays by allowing frames to be sent over the shared medium without interference from other senders.

As mentioned in Section 2.3, there are two kinds of ad hoc networks: MANET and WSN. This section describes the MAC techniques for MANETs and there are three types:

- Contention based (continuous time);
- Allocation based (discrete time);
- Hybrid approaches that try to combine the best features of contention and allocation techniques.

### 3.3.1 Contention Based MAC Protocols

Contention based MAC protocols can be classified into four categories [37]:

- No coordinator: nodes transmit at will when they have data to send (e.g., ALOHA [44]);
- Carrier sensing: nodes listen to the channel before transmitting a data packet (e.g., CSMA [45]);
- Carrier sensing and collision detection: nodes listen before and during transmission, and stop if a collision, that is, noise, is heard when transmitting (e.g., CSMA/CD [46]);
• Collision avoidance: a handshake is typically used to determine the node that can send a data packet (e.g., MACA, FAMA, IEEE 802.11, CSMA/CA, RIMA, and many others [47, 48]).

Contention based protocols achieve high throughput with a reasonable expected delay, in the worse case delay is very poor. These protocols rely on probabilistic delays.

3.3.2 Allocation Based MAC Protocols

In order to obtain a deterministic delay it is necessary to divide the signal into different time slots and allocate each slot to each transmitting node (TDMA) [49]. The use of TDMA provides a delay bound but it does not meet a reasonable throughput objective because it does not take advantage of spatial reuse in MANETs.

Because of poor spatial reuse, it is not common to find only allocation based protocols. Normally it is more an hybrid approach where contention and allocation is combined together.

3.3.3 Hybrid MAC Protocols

The hybrid approach combines a contention phase in which neighbor information is collected with an allocation phase in which nodes transmit according to a schedule constructed using the neighbor information.

Topology-dependent protocols (also called spatial used TDMA) use this technique of alternating the periods of contention and allocation [50, 51].

Figure 3.6 shows the different types of MAC protocols for MANETs described above.
3.4 WSN MAC Protocols

The other type of ad hoc networks is the WSN. The MAC protocols described for MANETs are usually not suitable for WSN because they focus on two important performance metrics: throughput and latency. However, these metrics are of secondary importance for WSNs. For WSN the energy consumption is of primary importance. Hence, the MAC protocols for WSN need to be developed according to this challenge [52].

There are three types of MAC techniques for WSN:

- Contention based;
- Reservation based;
- Hybrid solutions.
All these techniques depend on two fundamental multiple access schemes: Carrier Sense Multiple Access (CSMA) and TDMA.

Figure 3.7 shows the different types of MAC protocols for WSN.

![Figure 3.7: MAC Protocols for WSN [52]](image)

### 3.5 Formation Network Techniques

The MAC techniques described in Sections 3.3 and 3.4 are commonly applied in formation network techniques [53, 38].

In a multi-hop network, a common way to transmit information between a group of robots is by broadcasting the information. The authors in [54] state that these broadcasting protocols rely on simplistic form of broadcast called *flooding* wherein each node re-transmits each received unique packet exactly one time. Typically flooding results in unproductive and potentially harmful bandwidth congestion and inefficient use of node resources. To overcome the flooding problem, some current
broadcasting protocols present more efficient broadcast techniques whose goal is to minimize the number of transmissions while attempting to ensure that each packet sent is delivered to each node in the network. One of the most common methods are based on the TDMA channel access method.

In TDMA, several nodes share the same communications channel by dividing it into different time slots. Each node may only transmit during its assigned time slot. With this, each node is using only a part of the channel allowing multiple nodes to share the same transmission medium in a structured way that eliminates interference between the nodes.

Applications of TDMA protocols are prevalent in literature:

In [55], the authors propose RT-Link, a time-synchronized link protocol. The main goal of this protocol is to save energy (low power consumption) and it relies on an external clock pulse to synchronize the nodes. The major disadvantage of this approach is the use of a global clock to synchronize the nodes.

In [25], the authors propose an TDMA-based wireless communication protocol for teams of mobile robots where the size of the TDMA frame is adaptable. The TDMA frame size is not fixed and can change according to the desired number of transmitting robots. The protocol is event-based, however the robots need to be synchronized with a central agent.

In [56], the authors introduce a new concept of broadcast communication that they call omnicast. Omnicast is similar to a concurrent multiple broadcast from every node to every other node. They also study how a TDMA scheduler can optimize the release of information for an omnicast communication.

In [23, 24, 26], the use of a TDMA protocol to schedule the robots within a group is suggested while CDMA protocol is implemented in [27].
Table 3.2 summarizes the formation network techniques described above.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Scheme</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony [55]</td>
<td>Time synchronized link protocol</td>
<td>Major disadvantage is the use of a global clock to synchronize the nodes</td>
</tr>
<tr>
<td>Santos [25]</td>
<td>TDMA - Based</td>
<td>The protocol is event-based, however the robots need to synchronize with a central agent</td>
</tr>
<tr>
<td>Ram [56]</td>
<td>Omnicast (Similar to Broadcast)</td>
<td>The author also tested a TDMA scheduler</td>
</tr>
<tr>
<td>Thomas [23]</td>
<td>TDMA</td>
<td>There is no connection between the MAC protocol and the actual device used to transmit the messages</td>
</tr>
<tr>
<td>Felix [26]</td>
<td>TDMA</td>
<td></td>
</tr>
<tr>
<td>Ergen [24]</td>
<td>TDMA</td>
<td></td>
</tr>
<tr>
<td>Ho [27]</td>
<td>CDMA</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Formation Network Techniques

### 3.6 Time Synchronization Techniques

The majority of the network formation techniques mentioned in Section 3.5 are time-based, i.e., they rely on global clock synchronization to coordinate the data transmission between nodes.

The traditional synchronization schemes for wired networks, such as the Network Time Protocol (NTP), were designed for large-scale networks with a relatively static topology. These protocols assume a fully-connected or low latency topology, where any node can send a message directly to another node at any point in time in a single-hop fashion. This means there is a constant latency and jitter bound for all
messages in the system, and a close approximation for the actual latency can be provided [57].

To implement NTP in an ad hoc network, there would have to be a single master node, which would be problematic because the co-operating nodes might end up using different synchronization paths. This would result in different timing offsets with respect to the master node. Additionally, using a single master node makes the network vulnerable [58].

Another problem in a MANET or WSN is that hardware and software maintenance is generally less frequent or in some cases not possible due to physical location of the nodes or the simple fact that it is not possible to manually configure such a large number of nodes. In NTP applications, normally there are lot of nodes, but at the same time, human intervention is necessary for a certain group of nodes. This aspect is not necessarily portable to ad hoc networks [57].

3.6.1 Ad Hoc Synchronization Protocols

As described before one of the main challenges of ad hoc networks is the effect of broadcast in a wireless channel. Since synchronization can only be achieved through communication between nodes, the effects of the wireless channel need to be carefully considered in the design of synchronization protocols. However, wireless communication introduces randomness in the delay between nodes [58].

Many synchronization protocols for ad hoc networks aim to minimize the effects of this random delay.

Figure 3.8 shows some existing synchronization protocols for WSN and below it, it is the respective protocols description.
Figure 3.8: WSN Synchronization Protocols [52]

- Timing-Sync Protocol for Sensor Networks (TPSN);
- Reference-Broadcast Synchronization (RBS);
- Adaptive Clock Synchronization (ACS);
- Time Diffusion Synchronization Protocol (TDP);
- Rate-Based Diffusion Protocol (RDP);
- Lightweight Tree-Based Synchronization (LTS);
- TSync: hybrid protocol that combines LTS with RBS.

### 3.7 Heuristics for Communication Delays

In the previous Sections it was presented the inclusion of communication models in the feedback loop controller or the use of MAC protocols to allow frames to be sent over the shared medium in order to minimize communication delays.

Another solution relies on using heuristic methods techniques to allocate and schedule the communications [59]. In this area of research the communications are
approached as a task allocation and scheduling problem. In [60], the authors present a survey about scheduling in distributed systems for various software and physical architectures models.

The above problem of allocating and scheduling tasks in distributed systems is computationally complex (NP-hard) [59], so the use of heuristic methods is an advantage to find an acceptable solution in a reasonable time.

Some work in this area either assumes no communication delay at all or it considers a constant overhead [61, 62, 63, 64, 65].

Other approaches consider the delays due to an effective medium access control protocol [66, 67, 68]. They are mainly based on CSMA/CA and TDMA medium access control protocols.

Regarding the heuristics methods applied, it can be used like list algorithms [62, 65], clustering algorithms [59, 61, 64] or non guided search heuristics like tabu search, hill climbing, simulated annealing [69, 70, 68] or genetic algorithms [59, 71, 72].

In all the methods, the medium access control protocol used has an influence on the allocation algorithm performance. Nevertheless heuristic methods need to be aware of the local minimum problem.
Chapter 4

Proposed Architecture

This Chapter describes requirements of the proposed controller architecture to be used for multi-robots systems. Section 4.1 describes the main delay components present in a communication path. Section 4.2.1 and Section 4.2.2 present the required features to be present in a microcontroller, communication / time synchronization and communication protocol in order to minimize delays, respectively. Finally the overall architecture is described in Section 4.3.

4.1 Introduction

As mentioned in Chapter 1 (Section 1.1), it is important to study and analyze techniques or solutions that can minimize network delays.

The four main components that contribute to the communication delay are referred to as the critical path in synchronization. The critical path is non-deterministic in nature and, hence, it creates a major challenge in exploiting the traditional offset estimation methods [52].
Considering Figure 4.1, the communication delay between two nodes has 4 components [52]:

- Sending delay \( (t_{send}) \)
- Access delay \( (t_{acc}) \)
- Propagation time \( (t_{prop}) \)
- Receiving delay \( (t_{recv}) \)

![SYNC](image)

**Figure 4.1: Synchronization delay between a pair of nodes [52]**

The sending and receiving delays \( (t_{send} \text{ and } t_{recv}) \) are related to the technology used (type of microprocessor) to process the information received or sent over the communication channel. The access delay \( (t_{acc}) \) is related to the time needed to access the transmission medium. The propagation delay \( (t_{prop}) = \frac{d}{c} \), where \( d \) is the distance and \( c \) is the speed of light, is the only delay component that it is not possible to include in the proposed architecture.

The rest of this Chapter explains what are the necessary requirements to be present in a controller architecture in order to minimize these communication delays.
4.2 Requirements

4.2.1 Microcontrollers

Mobile robots normally use microprocessors or embedded systems to handle the integration between sensors, actuators, computing elements and communication modules. Moreover, these embedded systems handle multiple tasks running simultaneously (e.g., Figure 4.2). This can be a time processing consuming process and it can introduce unpredictability.

![Figure 4.2: Example of multiple tasks that can be running simultaneously in a microcontroller](image)

On modern operating systems, threads are used to implement concurrency in the context of a single process. The interface between the operating system thread management functions and user programs is abstracted by means of a software li-
library, such as the well-known Pthreads library. Programming thread interactions in this manner has two main disadvantages: (1) threads are managed by the operating system and for each library function call the process context has to be switched between user and kernel mode; and (2) general-purpose programming languages used with thread management libraries do not prevent the developer from making critical mistakes in accessing shared resources and coordinating threads that may lead to race conditions or deadlock. Furthermore, even in the context of RTOS where software is written to minimize latency, there is always a limit on the minimum time necessary by the system to react to external events because of the overhead introduced by the multitasking system software execution.

The set of requirements to be present in a microcontroller is the following:

- Low level I/O peripherals interaction (SPI, \(I^2C\), 1 wire protocols, Universal Asynchronous Receiver/Transmitter (UART), Pulse-width Modulation (PWM), etc.);
- High level intelligent behavior;
- Fast real-time processing;
- Multi-tasking;
- Deterministic;
- Parallel / concurrent feature.

The key advantage of the application of a microcontroller with the above requirements is the concurrent, asynchronous nature of typical multi-robot systems equipment. The computing paradigm problem is exacerbated by the recent introduction
of inexpensive sensors with high sampling frequencies. Taking full advantage of these technologies requires the use of increasingly parallel algorithms. Perhaps most importantly, deterministic parallelism is a top priority as discrete control algorithms (based on the Z-Transform) almost always require a constant sampling time.

4.2.2 Communications and Protocols

As mentioned in section 3.5, a team of mobile robots is normally modeled as a mobile ad hoc network where communication between the nodes can be achieved through broadcast. A common issue with ad hoc networks is the latency introduced due to media access conflicts or packet loss due to interference that has a significant impact on the overall performance of the protocol.

From the Open Systems Interconnection (OSI) reference model for communication systems [53], the Data-Link layer (second layer) is responsible for finding a means for different senders to share the transmission medium. A sub-layer of the Data-Link layer, MAC, is responsible for allowing frames to be sent over the shared medium without interference from other senders. There is common agreement about the need to synchronize and coordinate the communications in order to achieve low data collision, high channel utilization and high performance.

Another feature that is normally neglected is the relation between the communication protocol and the hardware used to transmit the data (physical layer on the OSI model). There are physical limitations, for example on the number of nodes that can transmit using a certain physical layer that can limit the amount of nodes considered on the communication protocol.

The set of requirements to be present in a communication protocol is the following:
• Deterministic access to the transmission medium (low data collision);

• Synchronized communications;

• The protocol should be related with the hardware used to transmit the data.

4.3 Architecture Layout

Figure 4.3 summarizes the architecture requirements for the microcontroller and communication protocol and the respective delay component that it is trying to minimize.

The following chapters describe the proposed solutions, from the microcontroller to a communication /synchronization protocol, in order to minimize the delay components described above.
Figure 4.3: Architecture Requirements
Chapter 5

Technology Selection

This section describes the technology selected to implement the proposed architecture. It starts with a brief technology history in Section 5.2. In Section 5.3, it is described in detail the technology selected followed by the description of existing technologies in Section 5.4. In Section 5.5, it is described the conventional benchmarks; is presented the proposed methodology; and finally it is discussed and analyzed the benchmark results.

5.1 Introduction

When developing intelligent multi-robot systems, it is a major challenge to mix low-level sensor, actuator and communication tasks with high-level, mission-specific computation. The low-level tasks are critical for safe operation of the vehicle and have hard real-time requirements. The “intelligence tasks” are equally critical to achieving a successful mission and must also be executed efficiently and at a high priority. Current approaches to implement multi-robot systems focus primarily on
the low-level functionality, and they often do so using multiple microprocessors bound together with glue logic and yet another set of low-level I/O routines [39].

In the following sections, it is described a new technology for multi-robot systems that leverages a microprocessor technology stack sold by XMOS Ltd\(^1\). This technology enables a new approach toward the design of multi-robot systems by gracefully fusing the low-level I/O tasks with high-level intelligence tasks.

### 5.2 Brief History

The XMOS technology is based conceptually on the idea of transistor computers, AKA “Transputers,” which were the first microprocessor designed specifically to be used in parallel computing systems. The architecture of a Transputer is rather simple – instead of having a complex processor, a *transputer* consists of a family of chips. A Transputer can be seen as a microcontroller that can boot itself, it has its own RAM, serial bus, embedded real-time OS that can be used to perform complex I/O protocols in an efficient way [73].

A Transputer-based parallel processor is programmed using the OCCAM language, an adaptation of Hoare’s CSP [74]. When introduced (1984), the OCCAM programming language was quite an advanced concept, since it permitted explicitly parallel algorithms programming with fine grain parallelism (more than one instruction running in parallel), as well as inter-process communication using the CSP’s channel concept. Because OCCAM implemented the CSP concepts, its programs could undergo a formal analysis for correctness [75].

\(^1\)http://www.xmos.com/
At that time the Transputer technology was only applicable to special parallel machines that ultimately failed to successfully compete against mainstream technology that had initiated the CPU performance race. In the marketplace, it was perceived as being easier to use the well-developed sequential algorithms at higher clock rates, rather than parallel algorithms on dedicated, but slower, machines. Additionally, the transputer did not use a traditional operating system; the OCCAM program itself directly supported process management, communication and synchronization. Further difficulties included the hardware interface and the serial channel implementation was not well adapted for implementing reconfigurable communication networks between processors (only 4 channels per CPU were allowed).

Recently, the concepts introduced by Transputer computers have emerged in modern form under the brand name XMOS. The main goal of this new technology is to provide a highly parallel platform for implementing real-time applications. The XMOS architecture consists of monolithic, multi-core, multi-threaded processors (XCores) designed to execute several real-time tasks. Some concepts used by XMOS technology are based on the Transputer technology, such as: the communication links between cores (channels), the multi-threaded, multi-core processor, the ability to implement complex low-level I/O protocols (e.g. an Ethernet controller and PWM modulation). In addition, each core has its own memory. Finally, the communication between processors is no longer as restricted as it was on Transputers.
5.3 XMOS

The XMOS processors (XCores) are event-driven and not dependent on the traditional interrupt scheme. Each XCore can provide up to 400 MIPS for deterministic real-time tasks.

Figure 5.1\(^2\) depicts the design of each XCore, which has up to 8 threads, each having access to 16 dedicated registers and 64KBytes of shared SRAM. Hard real-time scheduling is guaranteed regardless of the state of other threads being serviced by an XCore. Additionally, each XCore has up to 64 I/O pins that can provide 10ns timing resolution. This allows for the creation of software-defined peripherals (e.g. SPI, UART, PWM, etc). Finally, the XMOS platform is inherently scalable since multiple XCores can be chained together seamlessly.

![Figure 5.1: XMOS Architecture (from www.xmos.com/)](image)

From a software development perspective, the XMOS specific logic that enables concurrent tasks, real-time control, hardware timers and ports is programmed in

\(^2\)www.xmos.com/
using the XC language. Essentially, XC provides the ability to combine clean, high-level modules with efficient and safe hardware I/O. Additionally, the platform is easily accessible to competent systems programmers. Logic and basic general-purpose algorithmic code are implemented in C/C++ and can often be imported directly into the XC environment.

The advantages of this form of scalability become manifest in real-time robotics applications. As multi-robotic systems become more popular, the need for vehicle platforms to perform outside of their initial design parameters also increases. This necessitates an architecture that allows for the easy integration of additional sensors, actuators, and other components.

### 5.4 Existing Technologies

While there is a wide variety of multi-robotic systems, only a few utilize the XMOS technology. For example, in [76] and [77], is used Microchip microprocessors, the PIC16F877 and PIC30F6014A, respectively. In [78] and [79] the ARM architecture is used and in [80], the system uses the PC104 Intel 80486 with a DOS-based application. In addition, there are other technologies that combine diverse processors. For example in [1] an x86 mini-ITX motherboard and a servo switch controller from Microbotics are used. In [81] the authors combine a micro-Linux computer (Gumstix) with an ATmega microcontroller. In [82] an embedded computer called Qwerk is used to perform the wireless and camera interfaces and a Gumstix Robostix is used to integrate sensors and perform the low-level control.

All these technologies, as mentioned before, are limited in the number of inputs and outputs that they can handle once the sensors and actuators are chosen; there
are also limitations when handling multiple tasks in real-time. Additionally, all of the
aforementioned systems utilized some kind of embedded real-time operating system
to provide multitasking and interrupt management. In this programming environ-
ment it is difficult to debug the entire system, and to guarantee system reliability.

5.5 Benchmark

The main goal of this section is to describe a new methodology to evaluate the I/O
responsiveness of microprocessors. This addresses the need for benchmarks for real-
time systems that measure critical properties for system design that are not currently
handled by traditional performance benchmarks. The benchmark developed under
this methodology is tested on three microprocessor architectures: ARM, PIC and
XMOS.

The obtained results also support the selection of XMOS.

5.5.1 Introduction

As mentioned before, embedded systems have multiple computational tasks run-
ning and interacting simultaneously.

Commonly, embedded systems require software to respond to many inputs and
events within tight time constraints (in “real-time”). RTOS are designed to manage
the execution of this software in embedded systems. Applications are controlled by
the RTOS, and their allocated CPU time is scheduled by the RTOS kernel. Typ-
ically, RTOS consist not only of a real-time kernel but also higher level functions
(applications).
In general, applications are composed of a set of tasks (threads of execution), some of which are designed to handle asynchronous events (interrupts) from the outside world. In conventional embedded systems, these tasks will run in a concurrent way (multi threads) on a sequential processor. The RTOS provides a multi-threading abstraction, which attempts to manage the allocation of processing and other resources across multiple concurrent threads and according to their performance/deadline requirements.

Embedded systems have considerable performance requirements for the tasks that they execute, and management of such systems can be equally onerous. Even though typically heavily optimized, the traditional RTOS enabled programming model introduces substantial unpredictability (due to interrupts) and inefficiency (due to context switching).

Embedded benchmarks should be designed to measure the performance of highly parallel, experimental, non-standard architectures. Ideally the benchmark should be flexible enough to work with unconventional processors and custom hardware implementations in order to provide a fair comparison of a wide range of architectures.

This section describes a new approach to attempting to characterize a single, but key, property of real-time systems: the speed at which a system can respond to an external stimulus. Furthermore, it tries to characterize how this property varies when the system is also performing many tasks that require responsiveness.

While characterizing a single property will not tell a system designer everything they need to know about a system, it is the author’s belief that this property reflects critical characteristics of a system for a range of real-time embedded tasks.
5.5.2 Conventional Benchmarks

Some of the most well known embedded systems benchmarks available are: EEMBC, MiBench, MediaBench, PapaBench, and Jembench [83, 84, 85, 86, 87]. The author in [88] groups these benchmarks in three types that are normally used to assess performance: synthetic, application and derived benchmarks.

Synthetic benchmarks measure the performance of individual components of a computer system, normally by exercising the chosen component to its maximum capacity (e.g. Dhrystone, Whetstone, etc). Application benchmarks, also called “real world” benchmarks, use code from real algorithms of full applications. Derived benchmarks (or “algorithm-based” benchmarks) are a combination of synthetic and application.

Besides the benchmarks already mentioned, the author in [89] presents a more detailed description and list of available embedded systems benchmarks. These benchmarks share some common problems such as: some of them do not represent applications that would run on an embedded processor; validating the results can be expensive; some benchmarks do not allow manual source code optimization. They are intended only for platforms with full compiler support; multiple interacting tasks (as present in real world embedded applications) are not covered by application benchmarks; and, some benchmarks are limited to Intel x86 architectures running UNIX compatible operating systems and do not support non-standard or experimental architectures.

A well known reference and industry standard benchmark for evaluating embedded microprocessors is the EDN Embedded Microprocessor Benchmark Consortium (EEMBC) benchmark suite\(^3\).

\(^3\)http://www.eembc.org/
According to the author in [88], “In the 1990s, the new innovations in hardware architecture drove a need for benchmarks based on real applications... EEMBC’s goal was to bring real-world application-based benchmarks to the world.”

As in the 1990s, innovations in hardware architectures are happening all the time. The question that arises is: how suitable are the traditional embedded benchmarks (that rely on interrupt routines) if non-traditional architectures (that do not rely on interrupts) are used as embedded hardware platforms? In other words, how representative are the results of such benchmarks for non-standard architectures?

Embedded benchmarks should be able to measure the performance of highly parallel, experimental, non-standard architectures and compare them to traditional interrupt-based ones. Ideally the benchmark should be flexible enough to work with both conventional processors and custom hardware implementations in order to provide a fair comparison of a wide range of architectures.

**XBS Benchmark**

In [89], the author presents a benchmark for non-standard architectures (XBS benchmark). The XBS framework is organized into 3 hierarchical levels (Figure 5.2a): *Kernels* - represent small compute-intensive pieces of code that perform a single algorithm. Examples include FFT, matrix multiply, Quicksort, DCT, RSA encryption, and image convolution; *Applications* - combine multiple kernels to perform an identifiable task such as perform JPEG compression of an image or decode an MP3 audio file; *Scenarios* - combine multiple simultaneously running applications to measure the performance of processors on typical workloads.

Although XBS presents a benchmark framework for non-standard architectures, it does not fully cover some important features that are important to measure with
benchmarks for embedded systems. For example: it does not include I/O interaction or any kind of interaction with the real world; It does not address how workloads will affect applications that have I/O timing constraints; It does not include communication or synchronization between applications.

The proposed benchmark framework, explained in the next section, is presented to extend and improve the XBS benchmark framework in order to test a wide range of different embedded architectures for real world applications.

5.5.3 Proposed Methodology

The proposed methodology is inspired by the work developed with unmanned ground and aerial vehicles in the University of Denver Unmanned Systems Research Institute (DU2SRI) lab, but it is significantly abstracted from the types of tasks we typically use in an attempt to be more generally applicable.
Based on literature review, the author believes that existing benchmarks are not capable of accurately measuring the specific characteristics of several modern microprocessors that are relevant to sophisticated, real-time embedded systems.

Moreover the authors in [90, 91] also state that researchers must describe modern computer usage in terms of scenarios consisting of numerous I/O streams, timing information, and parallel tasks that enter and leave the system, rather than in terms of programs executing in isolation from the physical world and each other.

Most of embedded systems benchmarks tend to test processor core throughput and the core’s interaction with the memory.

This is due to the lack of I/O interaction as an integral part of the benchmark. Instead, the I/O features may be used to trigger benchmark execution and are used to record the results. Both actions are typically not recorded as part of the benchmark execution time. As an example, the NXP LPC2103 ARM7TDMI processor [92], performs the communication between the processor core and other peripherals (e.g. general purpose I/O (GPIO), Analog to Digital Converter (ADC)) by means of two bus bridges; each bridge adds latency to the I/O interaction. The LPC210x series is an interesting example of the shortcomings of typical benchmarks because the microcontroller series has two GPIO peripheral registers controlling the same set of I/O pins. The first set of registers is used for high speed interaction and the second set is used for normal speed interactions. Square wave generation tests reveal the significant disparity between the two communication methods with the fast GPIO achieving speeds up to 3.5x that of the normal GPIO peripheral [92].

The proposed benchmark is organized along two orthogonal task groups. The first task group, the “characteristic tasks” represents the particular capability of the system that is being measured. The second task group represents the “workload
tasks” and it is intended to provide the background functionality that it is the context of measurement (Figure 5.2b).

The “characteristic task” will be implemented to have a continual, measurable, interaction with the outside world via the I/O pins of the device under test. These interactions are monitored and logged with an oscilloscope and provide the data to compare different microprocessors architectures.

**Characteristic Task**

The key property of the benchmark in this thesis is the I/O response latency. This can be characterized relatively simply: The response latency of a system is the time it takes the system to change an output signal in response to a change in an input signal.

To measure this response latency the device under test is connected to an oscilloscope and a signal generator through two wires: the input wire contains the signal to be received by the device under test and it is connected to the signal generator; the output wire contains the signal to the output by the device under test and it is connected to the oscilloscope.

The test consists of the generation of a square wave. At this point the device under test should react and drive the output signal according to the square wave. The oscilloscope shows the time between the driving of the input signal and the change in the output signal. Figure 5.3 shows the interaction of the two signals and the measured response time.

The oscilloscope used to measure the response time has 4 channels with 200MHz of bandwidth and 2GSPS of sampling rate (TEKTRONIX - TDS2024C).
The waveform from an oscilloscope is set in persistent mode in order to acquire the best and worst case response time. The results are saved to a USB disk from the oscilloscope.

**Workload Task**

The workload tasks model the idea that in a real application the system will be doing many things at once. What the task is and how it is implemented depends on the system. These tasks are intended to be abstractions of real tasks that will be encountered across a wide range of application areas, such as: compute/intelligence/data fusion tasks - image manipulation (LCD/camera), graph algorithms, classic CPU/memory tests (e.g., use of Coremark from EEMBC, etc), encryption algorithms; Input-only tasks - pin monitoring, UART read, sensor reading (unidirec-
Output-only tasks - PWM, pin actuation, UART write; Bidirectional tasks - communication protocols such as I2C, SPI, UART, CAN, etc.

Overall, the functionality of the task itself is not that important. Generally a simple loop that performs some calculation and some reads and writes from memory will suffice.

**Response Time**

The response time of a system may vary depending on its state. The jitter of the response time is the extent of this variation (i.e. the maximum observed response time less the minimum observed response time) (Figure 5.3).

A sensible benchmarking strategy is to run the test multiple times and keep track of the average, minimum and maximum response time. In this case this is done by setting the oscilloscope in persistent mode. With this mode on, it is possible to acquire the best and worst response time as shown in Figure 5.3.

**Performance normalization**

As previously stated, the aim of this thesis is to use the benchmarking to compare system architectures rather than individual devices’ absolute performance. Clearly, a device running at a higher clock rate will generally run faster and it is more likely to get a better response latency. For the purpose of comparing architectures, it is necessary to normalize the clock frequency of the device.

There are two approaches to this: running every device under test at the same clock frequency or normalize by dividing the result by the clock frequency. The first approach is probably a more realistic test but it is not always possible due to the oscillators and clocking capabilities on specific devices. In the experiments presented
here a dual approach was taken: the clock speeds were chosen to be as close as possible to each other and then a normalization factor was applied to the results of each device.

5.5.4 Platforms Under Test

The following microcontrollers were selected for evaluation:

XMOS

SPECS: XK-1A development board at 80MHz. XMOS Development Tools 11.2.2. No RTOS.

- Part No.: XK-1A
- MCU clock: 80MHz
- Dev. Tools: XMOS Development Tools 11.2.2
- RTOS: hardware scheduler

The XMOS line of processors utilize an unique “concurrent processing” architecture and allows for multiple tasks to be executed without the use of a software implemented RTOS scheduler. Furthermore the IC lacks most general peripherals with the exception of a tightly integrated GPIO module.

PIC

SPECS: dsPIC33FJ256MC development board at 80MHz. MPLAB v8.80. FreeRTOS 7.0.
The dsPIC processor utilizes a DSP core communicating with the “W-register” system utilized by most Microchip PICs. This register then communicates with the GPIO peripheral via a 16-bit bus. As the processor core is single threaded, an RTOS is utilized for concurrency.

ARM


• Part No.: LPC1768-H
• MCU clock: 100MHz
• ARM Cortex-M3
• Dev. Tools: Keil uVision 4.21
• RTOS: FreeRTOS 7.0.1

The ARM processor’s internal architecture is similar to the the ARM7 explained before. As with the dsPIC, the ARM has no provision in hardware for parallel task execution and must utilize an RTOS for concurrent execution.
5.5.5 Evaluation

Results

The results were obtained by varying the number of workload task between 2 to 8 tasks. For all the experiments three response times were acquired: worst case, best case and jitter. Figure 5.3 from Section 5.5.3 shows how those response times are related to the signals used during the tests.

Figure 5.4 shows the normalized worst response latency of the three architectures benchmarked as the number of workload tasks increases. This might be a relevant measurement when designing hard real-time systems.

Figure 5.5 shows the normalized best response latency. Figure 5.6 shows the jitter, that is, the spread between best and worst case response time. The jitter measurement is important to get a feeling about the response latency “bandwidth” that is possible to achieve.

![Performance (log scale)](image)

Figure 5.4: Worst Case Performance Vs Number of Tasks (Normalized Results)
Figure 5.5: Best Case Performance Vs Number of Tasks (Normalized Results)

Figure 5.6: Jitter Performance Vs Number of Tasks (Normalized Results)
Analysis

Overall it is clear that the XMOS devices exhibit better characteristics in terms of response latency, in particular, the worst case latency and the jitter being significantly less than the other architectures. This is perhaps unsurprising given that the architecture is specifically designed for this type of task. The difference can mainly be attributed to the fact that in the XMOS architecture, each response is handled by a separate hardware thread which can respond to the input signal without needing a context switch.

Both the ARM and PIC architectures implement the response via interrupts (the lowest latency method available on those systems) which requires a context switch before response. The concatenation of these context switches explain the linear growth (logarithm growth in a log scale) in response time given the number of responses needed.

Every attempt has been made to make the comparisons in the previous section represent the best performing setups of the different architectures given realistic assumptions about how the processors would be used for real applications. However, there may be trade-offs designers can make that either negatively affect the response latency or may improve it in certain cases. On both the PIC and ARM chips, one of the key factors is the choice of real-time operating system. In fact, these benchmarks could be seen as characterising the combination of both the architecture and the RTOS. Different combinations will likely have different results. In some cases no RTOS will be used at all and that is likely to improve the results, though at the cost of having to hand craft the context switching, interrupt servicing and task scheduling within the system.
Even given the variations in results that these design trade-offs can make, it is the authors’ view that the architecture comparison presented here is representative of what the various architectures can achieve, and shows the value of an I/O response benchmark.
Chapter 6

Robots Scheduling

This Chapter describes the proposed communication protocol in order to allow synchronous communications among a group of multi-robots. Section 6.1 starts with a small presentation about the need for RT-MAC. Section 6.2 describes the requirements, frame description, scheduler process, behavior state machine and protocol analysis for RT-MAC. Then time-line examples are shown in Section 6.3. The simulation software and the performance metrics used to simulate and analyze RT-MAC are presented in Section 6.4. It is also presented the respective simulation and analysis of the results. Finally the the hardware implementation is described in Section 6.5.

6.1 Introduction

As mentioned before, a group of robots is normally modeled as a mobile ad hoc network where communication between the participants is achieved through broadcast. A common issue described earlier, with ad hoc networks is the latency
due to media access or dropped packets that has a significant impact on the overall performance of the group formation.

To achieve reliable formations the formation control needs to be distributed. One of the characteristics of distributed control is the guarantee of the packet delivery time. That means, the packets need to be received between a time interval, it is desired to be deterministic. This packet delivery time is achieved by using synchronization protocols. The synchronization protocols mention in Section 3 are not sufficient to overcome this problem. Synchronization protocols for MANETs are designed to achieve high throughput with a reasonable expected delay (some cases it uses probabilistic delays) and they are not deterministic. On the other hand, synchronization protocols for WSN are designed primarily for energy consumption.

As explained in Section 4, the critical path takes an important role when designing synchronization protocols. The critical path is non-deterministic in nature and many synchronization protocols for ad hoc networks try to minimize this random delay.

### 6.2 RT-MAC

The proposed solution is called RT-MAC, for Robot Team MAC. This solution differs from MANETs synchronization protocols because it does not rely on acknowledge packets and differs from WSN synchronization protocols because it does not have power restrictions.

The main idea of RT-MAC is to take the advantages of several different types of TDMA protocols and combine what we believe are the strong features of each into a novel event-based communication protocol. Figure 6.1 shows where RT-MAC is placed on the ad hoc networks protocols.
Figure 6.1: RT-MAC place in Ad Hoc Networks Protocols
RT-MAC is intended to be implemented in the second layer (data link layer) of the OSI model and it will work with any physical layer (e.g. WiFi or Very High Frequency (VHF)).

6.2.1 Requirements

RT-MAC has the following requirements/goals:

- It must be decentralized, without need for global clocks or other coordination;
- The nodes must be capable of transmitting constantly (i.e., there should be no power restrictions for wireless transmission - as is the case for mobile robots or vehicles where motors typically dominate the energy budget);
- It must support dynamic addition and removal of nodes from the group;
- It must be configurable for different sized message payloads; and
- It must be compatible with existing communications protocols (e.g. Micro Air Vehicle Communication Protocol - MavLink\(^1\)).

In order to satisfy these requirements, it is proposed an event-based scheme in which the nodes transmit in a TDMA fashion (i.e., only during their designated time slot), but where the time offsets in a frame are calculated locally by each node relative to the time of the previously received transmission. In order to support dynamic addition and removal of nodes, the frame has both a statically assigned part, and a dynamic part.

It is important to emphasize two aspects of the proposed scheme: (1) it is not suitable for high throughput data streaming, but rather it is designed for short control

\(^1\)http://qgroundcontrol.org/mavlink/start
and coordination messages (for high throughput data, a separate, dedicated channel would commonly be used); and (2) RT-MAC is not a routing protocol and it assumes direct connections between all nodes (fully connected graph).

### 6.2.2 Frame Description

In keeping with TDMA terminology, communication between nodes is broken into fixed-sized frames; within a frame, each node in the group has a designated slot in which they may transmit. The basic RT-MAC frame structure is shown in Figure 6.2. The amount of time available to each slot is known as the Data Transmission Time (DTT), where \( DTT = \frac{DTS}{MTR} \). The Data Transmission Size (DTS) is driven by the needs of the application layer, while the Maximum Transmission Rate (MTR) is determined by the device used for the physical layer.

![Figure 6.2: The RT-MAC Frame](image)

In real applications, transmission of data is not instantaneous. It is necessary to include a Guard Period (GP) that allows for slight mistiming in the transmission of the burst in the frame. The Frame Length (FL) is also a variable that can be changed according to the needs of the application and the group size. Therefore, for the simple RT-MAC frame, there are 4 variables (DTS, MTR, GP, FL) that
determine the maximum number of nodes allowed per group and the update rate for group data.

Obviously, communications are never perfect, and with the simple RT-MAC each node will transmit only once per frame. This introduces the risk that a node’s transmission is lost due to environmental interference. As each node re-calibrates its local clock upon message receipt, lost messages increase the possibility of significant clock skew.

There are at least two possible ways to overcome this problem: the use of acknowledgments for each packet transmitted; or the use of Packet Repetition (PR). We choose repetition, as acknowledgments in a broadcast scenario introduce unnecessary complexity, and introduce variability in latency which is what we are trying to avoid. Therefore, another variable parameter of RT-MAC is the number of packet repetitions per node, per frame, as shown in Figure 6.3).

![Figure 6.3: RT-MAC Frame with Packet Repetition (PR)](image)

Obviously, the effect of increasing PR is to decrease the number of nodes that can transmit on the RT-MAC frame. On the other hand, it increases the reliability of packet transmission without requiring explicit acknowledgments.
In Section 6.2.6 we analyze the number of nodes that can transmit in one RT-MAC frame for different values of tunable parameters (DTS, MTR, GP, FL and PR).

6.2.3 Scheduling

To assign slots to the nodes in the formation it is important to have some organization among them. In particular, it is necessary to choose a team leader to initiate a frame. The process of selecting a leader in a distributed system is known as leader election and the author in [93] describes some canonical leader election algorithms.

The scheduling (or slot assignment) in RT-MAC is done in a decentralized manner. It is envisioned a table (e.g., organized by unique identifier) that each node will maintain to indicate the organization of the group at any given moment. Knowing its own position in the table, each node in the group will transmit on the respective slot.

One solution for dynamic group membership is to have different types of slots in the same frame. Some examples of this implementation can be found in automotive network communications protocols such as Byteflight [94] or FlexRay [95]. These protocols use a hybrid synchronous/asynchronous TDMA-based method to allow communication between vehicle components. They were designed for high-speed, deterministic communications on wired networks. ByteFlight works with a global pulse to synchronize all components and it has fixed time slots for high priority communication while the rest of the bandwidth is allocated to low priority communication.

Similarly, in RT-MAC, slots are categorized as either static or dynamic. The static slots are used for the nodes that already belong to the formation and the
Dynamic slots are to transmit any type of information to the group (by members and nonmembers). As shown in Figure 6.4, the static slots are at the beginning of the RT-MAC frame and the dynamic slots are at the end of the RT-MAC frame.

![Figure 6.4: Frame with Static and Dynamic Slots](image)

FL is equal to the Static Slot Time (SSTime) plus the Dynamic Slot Time (DSTime). During the dynamic slot time the nodes will transmit information at random offsets to reduce the probability of interference. Adding this feature to the RT-MAC decreases the number of nodes that a group can have for a given FL, but it makes the protocol more generally useful. The impact of adding this feature to the RT-MAC frame is studied in Section 6.2.6.

### 6.2.4 Behavior State Machines

In order to build the table (e.g., organized by unique identifier) described in the previous section, it is necessary to combine the RT-MAC protocol with a behavior
state machine, and to add some information into the RT-MAC frame that will allow robots to join or leave the formation.

Figure 6.5 presents the Unified Modeling Language (UML) state machine that will allow each robot to build the table based on what it observes of its surroundings.

Upon initialization, a robot will monitor the environment, attempting to determine if an existing group is communicating in its area (LearnScheduler state). If there is such a group, the robot will attempt to join (JoinGroup state). If there is no existing group, the robot will assume the leadership role and begin transmitting frames and listening for other group members (CreateNew state).
Figure 6.6 presents the UML for the LearnScheduler and CreateNew, Figure 6.7 presents the UML for the JoinGroup and LeaveGroup and finally Figure 6.8 presents the UML for the GroupActive submachine states.

Figure 6.6: Submachine states: LearnScheduler and CreateNew

Figure 6.7: Submachine states: JoinGroup and LeaveGroup
Figure 6.8: Submachine state: GroupActive
6.2.5 Hybrid Group Control

One common and accepted approach to perform collaborative and cooperative behavior with multiple nodes (e.g., formation control) is to organize them into groups. The main property that all groups have is that when a message is sent to the group itself, all members of the group receive it.

There are two types of group control: centralized and decentralized each with well known advantages and disadvantages. RT-MAC is designed to support a hybrid group control in a decentralized way. This implies that all nodes in the group will have the same control logic but at any time, there is an organization among them. The organization makes the group control centralized (there is a leader) but in the case of lost leader some other node can easily step into that role.

6.2.6 Protocol Analysis

This section presents the analysis for the number of nodes (N) that a RT-MAC frame can support for different values of DTS, MTR, GP, FL and PR. The RT-MAC with packet repetition (PR), (Figure 6.3) is used for analysis. Equation 6.1 describes the relationship between these variables.

\[
N = \left\lfloor \frac{FL}{(GP + DTT) \times PR} \right\rfloor \quad (6.1)
\]

Where \( DTT = \frac{DTS}{MTR} \).

For a particular application, the size of the payload and the specifications of the module used to transmit data are known. Therefore, for this analysis, DTT is fixed and equal to 2.104 ms = \( \frac{263 \text{ bytes}}{1 \text{ Mbit/s}} \). The value of 263 bytes is the maximum packet length for the MavLink communication protocol. MavLink is an open source, lightweight,
header only message marshaling library for micro aerial vehicles. It has been used by several platforms (e.g., ArduPiloMega\textsuperscript{2} and UC Santa Cruz SLUGS\textsuperscript{3}) and represents the current state of the art for autonomous vehicle and robot communication.

The MTR is dictated by the capabilities of the physical device used to transmit data. One currently available commercial module used for these systems to transmit data is the WiFly GSX 802.11b/g Serial Module from Rover Networks\textsuperscript{4}. We select the slowest transmission rate of this device (1Mbit per second), and use this for analysis.

Equation 6.1 assumes that all slots are static. However, to support dynamic additions and removals from a group, there must be both static and dynamic slots. The inclusion of dynamic slots depends on the handling of GP, the guard period.

The GP is related to the propagation delay and it is challenging (if not impossible) to quantify the amount of wireless interference in a given environment and therefore determine the propagation delay. A possible starting place to estimate a value for GP in wireless environments is to use Propagation Delay = GP = $\frac{d}{c}$, where d is the distance and c is the speed of light.

Since it is hard to quantify a suitable guard period precisely, we analyze the situation according to the relative size of GP and DTT (the size of each slot) as follows:

\textbf{If GP} $\gg$ \textbf{DTT}

If GP is much bigger that DTT, then the solution relies on sacrificing the transmission time of one node and using those PR slots as dynamic slots. For example if

\textsuperscript{2}http://code.google.com/p/ardupilot-mega/
\textsuperscript{3}http://slugsuav.soe.ucsc.edu/index.html
\textsuperscript{4}http://www.rovingnetworks.com/products/RN_171
N = 8 then the maximum number of nodes allowed to transmit on the static slots would be 7 and the slots from the 8th node are scheduled dynamically.

In this case the dynamic slots will have the same size as the static slots (same DTS), and the maximum number of nodes that can transmit over the dynamic time is equal to PR. It is difficult to implement this feature without sacrificing some of the static slots. An immediate answer is to decrease DTS (smaller message size) and that will decrease the DTT time. But it is important to note that the dynamic slots are also dependent on the GP time and GP \( \gg \) DTT.

As an example, assume GP = 10 ms, DTT = 2.104 ms, PR = 5 and FL = 500 ms then N = 8. In this case GP is 475% higher than DTT with DTS = 263 bytes. As it is possible to see in Figure 6.9, with FL, GP, PR and MTR fixed, decreasing the size of DTS is irrelevant. So it is necessary to sacrifice static slots if GP is much bigger than the DTT.

**If GP \( \approx \) DTT or GP \( \ll \) DTT**

If GP is similar to DTT or smaller than DTT then one of the solutions relies on decreasing the size of DTS for the dynamic slots without jeopardizing the maximum number of nodes obtained with the remaining parameters. The maximum number of nodes that can transmit over the dynamic slot time \((N_{DS})\) is shown in Equation 6.2.

\[
N_{DS} = \frac{DS_{Time}}{GP + DTT_{DS}}
\]  

(6.2)

As an example assume GP = \( \frac{d}{c} \), with \( d = 100 \) m and \( c \) is the speed of light, DTT = 2.104 ms, PR = 5 and FL = 500 ms then N = 46.79. Without sacrificing static slots the solution is to use the remaining 0.79 node time from the FL as dynamic
Figure 6.9: N vs DTS (fixed FL, GP, PR and MTR)
slots. This means that \(((GP+DTT)\times PR)\times 46 = 491.51\) ms from FL is used for static slots and \(DSTime = FL - 491.51 = 8.49\) ms \((N_{RemainingTime})\) are for dynamic slots.

Figure 6.10 shows \(N\) for different DTS (fixed FL, PR, GP and MTR).

![Figure 6.10: N vs DTS for GP = DTT (fixed FL, PR and MTR)](image)

In Equation 6.2 the dynamic slot \(DTS_{DS}\) parameter \((DTT_{DS} = \frac{DTS_{DS}}{MTR})\) can be used to adjust the number of desired nodes to transmit over the dynamic slot time. Using the same example if \((GP+DTT_{DS}) = (GP+DTT)\) then \(N_{DS} = \frac{8.49}{0.033+2.104} \approx 3\).

Figure 6.11 shows \(N_{DS}\) for different \(DTS_{DS}\) (fixed \(DSTime\), GP and MTR).

### 6.3 Time-Line Examples

This section presents the time line diagrams for different examples of the RT-MAC communication scheme.
Figure 6.11: $N_{DS}$ vs $DTS_{DS}$ for (fixed $DS_{Time}$, GP and MTR)
All the examples are based on the RT-MAC frame without PR to better understand the communication scheme.

Figure 6.12 shows a basic time line diagram.

![Figure 6.12: RT-MAC Scheduler - Basic Example](image)

Slot color description:

- **Green**: leader static slot;
- **Dark blue**: normal static slot;
- **Light blue**: available static slot;
- **Light orange**: available dynamic slot;
- **Dark orange**: normal dynamic slot;

**Note:** Dark orange is not shown in Figure 6.12. It will be used later for joining and leaving formation examples.
6.3.1 Create Group

Figure 6.13 represents the time line to create a group of robots. There is a period where all robots broadcast the information and build the table (e.g., organized by unique identifier) according to their surroundings. After that period they start transmitting the information according to RT-MAC communication scheme.

![Figure 6.13: RT-MAC Scheduler - Create a Group](image)

6.3.2 Lost Leader / Assign New Leader

Figure 6.14 represents an assignment of a new leader.

![Figure 6.14: RT-MAC Scheduler - Lost Leader / Assign New Leader](image)

79
6.3.3 Join Group

For the join formation communication scheme it is presented 3 time line diagrams.

The first time line diagram (Figure 6.15) shows how one robot can join the formation. In this case robot 3 enters in a scheduler learning phase. Then it tries to join the group by transmitting on the dynamic slots time. It waits for a response from the leader and then successfully joins the group.

![Figure 6.15: RT-MAC Scheduler - One Robot Join Example](image)

The second time line diagram (Figure 6.16) shows how three robots can join the formation. In this example all robots first learn the scheduler by listening the environment. There are only available 2 dynamic slots for 3 robots. In this case the access to the dynamic slots is done randomly. Robot 5 and 3 successfully transmitted the request to join. Robot 4 transmits on the next time frame. In this case the leader accepts each robot at the time but it could accept all at once. At last robot 4 joins the group.

In this example the leader accepts each robot at the time to illustrate the transmission hierarchy on the static slots changing while the robots are joining the group.
The third time line diagram (Figure 6.17) shows a robot join failure. All the static slots are taken by 8 robots. Robot 9 attempts to join the group but the leader refuses the request.

Figure 6.17: RT-MAC Scheduler - Robot Join Failed Example

6.3.4 Leave Group

The time line diagram in Figure 6.18 represents one robot leaving formation. Robot 3 sends a message during the dynamic slots time communicating that it is leaving. The leader answers and the robot leaves the group.
6.4 Simulation

This section presents the simulation analysis of the RT-MAC scheme.

6.4.1 Simulation Software

The simulation results were obtained using Network Simulator 2 (NS2) [96]. NS2 is a discrete event network simulator widely used to simulate routing protocols and network protocols for wired and wireless networks.

The simulations have the following NS2 configuration:

- Topography of 500 by 500 meters;
- A two ray ground propagation model;
- Wireless channel;
- Wireless Phy (network interface);
- 802.11 MAC layer;
- Omnidirectional antenna;

Figure 6.18: RT-MAC Scheduler - One Robot Leave Example
• No routing protocol;

• Channel capacity of 1Mbps.

The nodes are not moving and they are positioned in a circle with a radius of 20 meters. The size of the transmitted packets is 263 bytes.

6.4.2 Setup and Performance Metrics

To analyze the impact of different transmission rates we created 4 experimental trials: (1) where GP >>> DTT and the overall slot time is 500 ms; (2) where GP >> DTT and the slot time is 10 ms; (3) where GP = DTT and the overall slot time is 4.208 ms; and (4) where GP << DTT and the slot time is 2.104 ms.

There are two sets of simulation results. The 4 trials were run for each simulation. For each trial the number of nodes is varied between 2 to 10 and run 100 times. The universe of study is 3600 results for each simulation. Each trial time is 190 seconds in virtual time.

The results are analyzed with two standard performance metrics:

• RSS: the ratio of the received packets to the sent packets for each node (Equation 6.3);

• LT: the link throughput, that is, the average of successful received packets during the simulation time (Equation 6.4).

\[
RSS(\%) = \frac{received\ pkts.}{sent\ pkts.} \times 100 \tag{6.3}
\]

\[
LT(bps) = \frac{(received\ pkts. \times pkt.\ size)}{simulation\ time} \tag{6.4}
\]
6.4.3 Results

For Figures 6.19, 6.20 and 6.21, the boxes show the 25%/75% boundaries of the distribution, the horizontal line shows the median, and the lines connected to the boxes show the 5%/95% boundary. The plus sign represents any outliers (i.e., individual points outside the 5%/95% bounds).

The first simulation consists of unordered transmissions, that is, all the nodes broadcast messages with no agreement about who is next to transmit the information. During the simulations, the nodes start transmitting at randomly selected offset times. Figure 6.19 and 6.20 shows the results for RSS and LT performance metrics respectively.

![Figure 6.19: RSS Results with unordered transmissions](image)

Figure 6.19: RSS Results with unordered transmissions
Figure 6.20: LT Results with unordered transmissions
In the second simulation, the nodes transmit messages in a round robin fashion, similar to how RT-MAC works. For this simulation the RSS value is 100% for all trials so we omit the figure. Figure 6.21 shows the results for the LT metric performance.

![LT Performance (Order)](image)

Figure 6.21: LT Results with ordered transmissions

### 6.4.4 Discussion and Analysis

In Figure 6.19, the RSS metric gives a good feeling for the importance of synchronous communications. With no order during transmission, when more robots are added to the group the RSS gets worse. The rate that the data is transmitted (GP+DTT) is also something that must be carefully chosen. Faster rates will have more impact on the RSS. In the 4th trial, the LT drop most dramatically with in-
creased nodes; this is due to the fact that $GP + DTT$ is smaller than the propagation time leading to more packets losses.

With ordered transmission, the RSS is 100%. Although the LT significantly less (compared with unordered transmission) synchronous communication among the nodes is achieved, therefore demonstrating the benefits that RT-MAC provides.

As expected, for both simulations the throughput is always impacted (Figure 6.20 and 6.21) by adding nodes into the group emphasizing the importance of considering both RSS and LT when designing a communication system.

### 6.5 Hardware Implementation

In this Section it is presented the hardware implementation of RT-MAC.

Figure 6.22 shows one RT-MAC node. It consists of one XMOS board and one WiFly GSX 802.11b/g serial module.

![RT-MAC Node with XMOS board and WiFly module](image)

Figure 6.22: RT-MAC Node with XMOS board and WiFly module

RT-MAC was implemented in the application layer using XMOS because the WiFly modules used don’t allow the implementation of code in layers below the
application layer. Figure 6.23 shows RT-MAC place in the OSI model using WiFi as the under layer.

RT-MAC was tested up to 5 modules in one group (Figure 6.24).

Figure 6.24: RT-MAC Nodes
Chapter 7

Communication Scheme

This Chapter describes the novel communication scheme that will enable simultaneous transmissions and it will allow deterministic communication delays. It starts with an introduction about the need for the proposed communication scheme in Section 7.1. Section 7.2 describes the existing work. The communication scheme analysis and algorithms implemented are presented in Section 7.3 and Section 7.4, respectively. The simulation results are analyzed and discussed in Section 7.5. Finally, the hardware implementation is addressed in Section 7.6.

7.1 Introduction

Network protocols for mobile robots rely on scheduling techniques (e.g., TDMA) to enable reliable communications. The majority of these protocols are time-based, i.e., they rely on global clock synchronization to coordinate the data transmission between nodes. They incorporate techniques to deal with the clock skew in order to achieve a common notion of time among all nodes.
A major concern with these protocols is the management of the shared transmission medium. For example, if multiple radios are transmitting simultaneously on the same frequency the signals may overlap, leading to interference. Wireless communications suffer from self-jamming as a consequence of multiple nodes attempting to transmit at the same time. The traditional solution to this problem is to use Request-To-Send (RTS) and Clear-To-Send (CTS) packets to claim the wireless medium for their transmission. The issue is that this solution introduces non-determinism as they wait a random delay before trying to transmit again.

In real-time systems, unpredictable delays are undesirable and numerous techniques have been developed to organize the behavior of transmitters to eliminate this problem. Existing techniques to eliminate “sense-before-send” rely on cooperation and coordination among all senders. All these schemes have something in common, they rely on two main techniques of coding theory including error detecting and error correcting codes to deal with corrupted data during a transmission [97]. These are data link control techniques to enable reliable delivery of digital data over unreliable communication channels. But they do not address the packet collision problem by themselves. They need to be combined with medium access control protocols to handle these ones.

The authors in [98] propose a new family of codes known as concurrent codes which are a form of superimposed codes. This approach enables a novel form of spread spectrum radio communication that can address some problems such as: jam resistance in public-access networks [99], Radio-frequency identification (RFID) self-jamming and MAC protocol simplification in wireless networks.

A concurrent code is a superimposed code that can be decoded in polynomial time. Furthermore, a concurrent code translates each message into a binary code-
word. The idea of concurrency comes from the ability to combine several codewords with a bitwise OR to form a single combined string. Then, the receiver can analyze the combined string and recover all of the original codewords and messages.

BBC (from the authors Baird, Bahn and Collins) is the algorithm that implements this encoding scheme permitting multiple messages sent concurrently to be received and separated.

7.2 BBC

BBC is fully described in [98, 99, 100]. Briefly, it consists of two independent stages: encoding and decoding. During the encoding stage, the algorithm accepts a binary string to be encoded and maps it to the proper transmission slot. The slot is just a conceptualization, and could be represented, for example, as specific frequencies, time slots, or any other form of distinguishable transmission. The decoding stage is responsible for extracting the messages contained in a sequence of transmission slots and supplying the originally binary string.

The author in [99] also states (up to that date) that BBC algorithm was the first system ever proposed to allow jam resistance communications without a shared secret key.

7.2.1 Existing Work

There are not many implementations of the BBC algorithm publicly available.

The original authors of BBC in [98] demonstrate the idea using sound waves instead of electromagnetic waves. The idea is to allow laptops to communicate using their built-in speakers and microphones, without adding external peripherals such as
Software Defined Radios (SDR). The demonstration ran with all 3 laptops sitting adjacent on a desktop, with the microphone of the receiver equidistant from the speakers of the other two. The tests worked well even in the presence of background noise and when the sending and receiving programs were run simultaneously on a single computer. The authors also mention that a new system is being built using SDRs.

Another implementation of the BBC algorithm was described in [101]. The author proposes a jam-resistance routing protocol for use in MANETs. The main advantage of the proposed routing protocol is that it adapts to the level of interference in the environment enabling effective communications. The author implemented and tested the routing protocol in two ways:

- Using a traditional 802.11 wireless communication system, implementing the code in a computer;
- Using an Universal Software Radio Peripheral (USRP) between 1150-1450 MHz, implementing the code using GNU radio software development toolkit.

A total of 10 nodes, 4 real physical nodes and 6 simulated nodes running on virtual network interfaces, were used [101].

Nowadays embedded systems are present in a lot of applications and more often these embedded systems must communicate. None of the known implementations of the BBC algorithm have been tested in real-time on embedded systems.

Implementing the BBC encoding on an resource constrained system presents significant implementation challenges, especially during the decoding stage.
Section 7.3 and Section 7.4 describe and evaluate the implementation of the BBC algorithm for an embedded system with the aim to use it as a clock synchronization technique.

7.3 BBC Analysis

7.3.1 Mathematical Foundations

Following the presentation in [98], the equations below describe the relationships among the maximum number of simultaneous messages, the bit rate, and the number of expected false-positive messages (known as hallucinations). These equations are used during the simulation (Section 7.5) to evaluate the performance of the proposed algorithms (Section 7.4).

Notation

- \( Ms \) = number of messages intentionally put into a packet by senders
- \( c \) = length of a codeword = length of a packet
- \( d \) = expected number of marks (1 bits) per codeword
- \( m \) = length of a message
- \( \mu_c \) = mark density (probability a codeword bit is 1)
- \( \mu_p \) = mark density (probability a packet bit is 1)
- \( h \) = expected number of hallucinations per packet

\[
\mu_c = \frac{d}{c} \tag{7.1}
\]
The equations above are derived based on an idealized random codebook, assuming the availability of unbounded memory and computational power for encoding and decoding [98].

### 7.3.2 Visual Representation

In Figure 7.1, a visual representation of a packet and codewords (messages to transmit) is shown. The packet is the logical OR result of the multiple codewords and it is what is transmitted on the shared medium.

![Figure 7.1: OR-channel result](image)

A packet can contain not only a group of messages that were initial encoded but also some additional messages. These extra codewords happen to have 1 bits in the same location as those in the original set. Two issues can arise during the decoding phase due to this extra 1 bits: hallucinations and repetitions.
7.3.3 Hallucinations

For the majority of superimposed codes, if the number of codewords is exponentially large, the decoding process will not be feasible. As mentioned in Section 7.1, BBC is a superimposed code that can be decoded in polynomial time. This is due to the fact that it only uses a subset of codewords of the codebook.

Even using a subset of codewords, the extra 1 bits in the packet described earlier can still make the decoder extract extra codewords that were never intentionally encoded. These additional false-positive codewords are the hallucinations.

An hallucination is an additional decryption message, that is, during the decoding stage the decoder algorithm extracted more messages than those that were transmitted. This is because the decoding method examines each possible codeword against the packet.

Reducing the number of codewords only reduces the probability of an hallucination happening. Hallucinations are always possible and they can be difficult to detect. BBC also reduces the probability of a hallucination by adding extra information to the message. The authors in [98] describe this as padding bits. The padding bits can be added to a BBC message to form a padded message.

There are 3 types of padding bits:

- Random Preamble Bits (R): these are random bits prepended to a message;

- Interstitial Checksum Bits (S): these are 0’s inserted at well-known positions into a message;

- Terminal Checksum Bits (K): these are 0’s that are appended to the end of a message.
The (R) padding bits will not reduce the probability of hallucinations. It will only enable an increase in the statistical independence when multiple messages are present. The (S) and (K) padding modes will manage the hallucination levels and kill hallucinations that survived during the decoding stage respectively.

If we see the decoding process as a decoding tree, adding S and K to the message will eliminate branches from the decoding tree. This is due to the fact that several consecutive 0’s are added at a specific location. If those consecutive 0’s are not received there is no need to continue to search through the decoding tree and the probability of a hallucination survival will be reduced.

7.3.4 Repetitions

A repetition can be thought of as an hallucination that can be detected and eliminated. A repetition appears when the receiver and transmitter are shifted by a multiple of the bit time (during the decoding stage) making it easy to detect.

Visually, from Figure 7.1, it means that the codewords do not match up in time.

The best way to understand a repetition is to consider Figures 7.2 and 7.3.

![Figure 7.2: Acquisition with No Repetition](image)
Figure 7.3: Acquisition with Repetition

For these figures a digital logic analyzer was used on real hardware to acquire the data and understand this issue.

The OR-channel result on Figure 7.2 is identical to the result on Figure 7.1, indicating everything is normal. On the other end, the OR-channel result on Figure 7.3 has some “spikes” that can lead to repetitions during the decoding stage.

These “spikes” are related with the BBC sampler task described in more detail in Section 7.6 and the fact that the codewords are not synchronized up in time. Due to the fact that transmitters are not synchronized with the receiver the BBC sampler task will be in sync with one of the multiple transmitters. For the senders that are not in sync with the receiver, there is a chance that the transmitted bits will show duplicated (i.e., be sampled multiple times). The “spikes” are then a symptom of duplicated bits.

### 7.4 BBC Algorithms

In this section the coding and decoding algorithms implemented for the simulation and hardware tests (Section 7.5) are presented. The algorithms described in this
section are the same for all the tests. Also, for both the encoder and decoder, the hash function \((H())\) used is the same and it is described in more detail in Section 7.4.3.

### 7.4.1 Encoder

The encoder method used in the coding stage is the same as the one proposed in [98] the “BBCbroadcast\((M)\)” algorithm.

**Algorithm 1 BBCbroadcast\((M)\) [98]**

This function broadcasts an \(m\)-bit message \(M[1 \ldots m]\) adding \(k\) checksum bits to the end of the message. \(H\) is a hash function. The definition of \(H\) and the values of \(m\) and \(k\) are public (not secret). The definition of “indelible mark” and “location” are specific to the physical instantiation of BBC used.

1. Append \(k\) zero bits to the end of \(M\)
2. **for** \(i \leftarrow 1 \ldots m + k\) **do**
   1. Make an indelible mark at the location given by \(H(M[1 \ldots i])\)
3. **end for**

### 7.4.2 Decoder

The decoding method described in [98] is recursive. A disadvantage of this decoder is not only the amount of memory that it is necessary to have available but it is also time consuming in a microcontroller. However, the main advantage is the versatility to decoding messages with different types of padding bits (explained in Section 7.3) without changing any code.

The state machine for our iterative decoder is presented in Figure 7.4. This method gives the same results as the recursive method. However, it accommodates the limited memory available in microcontrollers and improves the decoding time on the hardware we evaluated.
This is achieved by implementing the decoder with a state machine that “travels” along the decoding tree. The iterative method uses a stack to store the level of the decoding tree and uses a push and pop method to go up and down on the same tree. Once the stack is full there is a complete decoded message.

As mentioned above, the disadvantage of our iterative decoder over the recursive one is that it must be modified to accommodate different padding bits. For this thesis all the messages have only one type of padding bits, the terminal checksum bits (K). The proposed decoder algorithm is implement to accommodate this.

### 7.4.3 Hash Function

The selection of the hash function is an important step for the success of the BBC encoding. In the perfect hash function, by definition, every input value maps
to a unique output value (i.e., no collisions) [102]. In practice, a reasonable hash function will be the one that produces the least amount of collisions for a particular set of data.

There are different types of hash functions, from string hashing to cryptographic hashing. As mentioned earlier the difference relies in the compromise between speed and good hash values distribution. The hash function \( H() \) used during the coding and decoding stage needs to be implemented in a microcontroller. It is not necessary to encrypt the message, it should be computational light and have an acceptable hash values distribution. The hash function used in this thesis is the AP Hash Function\(^1\) which meets these goals.

### 7.5 Simulations

#### 7.5.1 Description

In this section, a simulation of the BBC protocol and the proposed algorithms is performed. The main goal of simulating the BBC algorithm is to study the impact of hallucinations and repetitions during communications in a controlled “environment” (no other sources of noise). We also use the simulation to implement realistic scenarios that help understand and debug the real hardware implementation.

For all the simulations in this section the following BBC parameters are used: \( c = 1000, d = 24 \) and \( m = 16 \) were used. \( Ms \) was varied for all tests from 1 to 50 transmitters (or nodes). These values were chosen to ensure that the simulated environment is as close to the hardware implementation as possible.

\(^1\) www.partow.net
Each message has the following structure: \(<\text{ID}(8\text{bits})>\ <\text{Counter}(8\text{bits})>\ <\text{K}(8\text{bits})>\)
that gives a total of 24 bits per message.

7.5.2 Results

Different configurations of transmitting nodes are evaluated. These ranged from all nodes transmitting synchronously to all nodes transmitting asynchronously. Noise was also added to study the impact of extra bits in the transmission medium. Table 7.1 summarizes the simulations performed.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>No Noise</td>
<td>all nodes transmitting with no noise</td>
</tr>
<tr>
<td></td>
<td>10% Noise</td>
<td>all nodes transmitting + 10% of random 1 bits in the packet</td>
</tr>
<tr>
<td></td>
<td>30% Noise</td>
<td>all nodes transmitting + 30% of random 1 bits in the packet</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Shifted</td>
<td>nodes transmitting shifted by 1 packet slot</td>
</tr>
</tbody>
</table>

Table 7.1: Simulation tests description

All nodes transmitting at the same time is the worst case scenario because the transmission medium has a maximum of 1 bits resulting from the union of all simultaneous transmitted messages.

As pointed out in Section 7.3, extra 1 bits in the packet can lead to hallucinations and repetitions during the decoding stage. To evaluate the impact of these issues the hallucination rate and number or repetitions were recorded for all simulations.

The transmission of 100,000 packets for each node was simulated. This corresponds to performing the same tests with the hardware described in Section 7.6 transmitting continuously for 41:36 hours.
Figure 7.5: Hallucination Rate vs #Nodes (Simulation)

Figure 7.6: #Repetitions vs #Nodes (Simulation)
7.5.3 Discussion and Analysis

The transmission medium is completely reliable for the results obtained with the simulator – it is a controlled environment, so all the packets sent are received. The unexpected received messages must therefore be hallucinations or repetitions and can be easily detected within the simulation.

The “Theoretical Boundary” curve in Figure 7.5 is obtained from Equation 7.3 under the assumption of an ideal random codebook and shows the expected worse case scenario. The expected simulated results seems that it should be under the “Theoretical Boundary” curve, but the 30% noise curve is above the theoretical curve. This is due to the effect of noise. Noise on the system similar to additional senders. So, essentially above a certain level of noise, the curve shifts to the left.

For the no noise curve, the ideal case, hallucinations start to happen when there are 29 nodes transmitting simultaneously.

Figure 7.6 shows the number of repetitions detected versus the number of transmitting nodes. This figure shows that as noise increases, so does the likelihood of a repetition.

7.6 Hardware Implementation

7.6.1 Description

In this section the implementation of the proposed algorithms with microcontroller hardware is presented. The processor technology used is the XMOS multicore multi-threaded microcontroller\(^1\). This technology enables the implementation of

\(^1\)http://www.xmos.com/
the encoder and decoder algorithms in a parallel fashion so both stages can execute simultaneously. In Section 5, it is described and analyzed with more detail why it was selected this microcontroller technology.

Figure 7.7 presents the tasks (conceptually similar to threads) used to implement a complete BBC communication system. Three tasks are running at the same time: BBCEncoder, BBCDecoder and BBCSampler. All the communications between tasks are done through XMOS channels.

![Figure 7.7: BBC Tasks](image)

The BBCEncoder task implements the BBCbroadcast(M) algorithm [98]. The BBCDecoder task implements the $iterDecode$ algorithm described in Section 7.4. The BBCSampler is a simple task that listens to the physical layer at the slot transmission rate and then transmits this sampled information to the BBCDecoder task. This allows the system to receive and decode several packets at the same time.

One important aspect to consider is the BBCDecoder executing time. It is important to guarantee that the time to decode the maximum number of messages present in a packet is less than the time of one packet bit slot time. That is, if the transmis-
sion of one slot of the packet takes $T$ ms then the BBCDecoder time shouldn’t take more than $T$ ms to decode the maximum number of messages available in a packet.

This raises the question how to select an appropriate value for the $c$ parameter from Section 7.3. During simulations, the physical layer from Figure 7.7 does not exist. The size of $c$ is not a big concern because the encoder does not have any interaction with the external world. For all situations, the parameter $c$ is related to the number of nodes simultaneously transmitting. However, in real hardware, the size of $c$ will also dictate how a node can transmit the information.

As an example, if the BBCEncoder task takes 1.5ms to transmit one slot then if $c = 1000$ then a node will take $1.5\text{ms} \times 1000 = 1.5$ seconds to transmit a message. At the same time with $c = 1000$ and a message with $d = 24$ codemarks, it is possible to have at least 41 nodes transmitting at the same time (discarding hallucinations and repetitions issues). So, the decoder algorithm must be able to extract 41 messages in the 1.5ms slot (bit) time.

7.6.2 Wired Test

Environment Description

In this section the BBC protocol was tested in a wired environment. As mentioned in Section 7.3, the packet is the logical OR result of multiple codewords. A channel over which multiple transmitters can broadcast their information in this way is called multiple access OR channel or simply OR channel [98].

For all the tests in this section the OR channel is simply an OR gate. All the transmitters are sending their information to an OR gate and the receiver is getting the result of it. Figure 7.8 shows the physical setup for the wired tests.
Results

Tests similar to those performed during simulation were done on the hardware. The main difference is that all nodes are transmitting asynchronously. Another board is used to add noise by randomly transmitting 1’s to the OR gate in order to study the impact of extra bits in the received packet. Table 7.2 summarizes the tests performed.

For the wired tests, the hallucination rate and number of repetitions were also recorded. Due to the limitation of available boards the maximum nodes used were 16 and each one transmitted a total of 800 packets. This corresponds to 20 minutes of continuous transmission.

Table 7.3 compares the results between the simulation and the wired tests for the maximum nodes (16 nodes) used during the wired tests.
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>No Noise</td>
<td>all nodes transmitting with no noise</td>
</tr>
<tr>
<td></td>
<td>10% Noise</td>
<td>all nodes transmitting + 10% of random 1 bits in the packet</td>
</tr>
<tr>
<td></td>
<td>30% Noise</td>
<td>all nodes transmitting + 30% of random 1 bits in the packet</td>
</tr>
</tbody>
</table>

Table 7.2: Wired tests description

<table>
<thead>
<tr>
<th>Type</th>
<th>#Nodes</th>
<th>Halluc.(%)</th>
<th>Rep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Noise Simulator</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10% Noise Simulator</td>
<td>16</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>30% Noise Simulator</td>
<td>16</td>
<td>0.67</td>
<td>0</td>
</tr>
<tr>
<td>No Noise Wired</td>
<td>16</td>
<td>0</td>
<td>4628</td>
</tr>
<tr>
<td>10% Noise Wired</td>
<td>16</td>
<td>0.04</td>
<td>3330</td>
</tr>
<tr>
<td>30% Noise Wired</td>
<td>16</td>
<td>951.5</td>
<td>7550</td>
</tr>
</tbody>
</table>

Packets Transmitted per node: 800
Packets Received per node: 800

Table 7.3: Simulation Vs Wired Results

Comments

The results were obtained in an uncontrolled environment for the wired tests. That is, there is external noise that cannot be detected or eliminated that will lead to extra 1 bits on the final OR gate. This extra 1 bits will contribute to the generation of hallucinations and repetitions.

When noise is not present or the level of noise is low (< 30%) the wired results are analogous to the simulations results with respect to the hallucinations. For the repetitions the results are not the same due to the fact that all nodes are transmitting asynchronously with respect to the receiver, as explained in Section 7.3.

When noise increases (> 30%) the hallucinations start to happen. The reason why hallucinations are not similar as the simulator is because the external noise that
is not purposely generated adds to the generated noise increasing the hallucinations and repetitions.

For all the results obtained with the wired tests there were no lost messages. In other words, while there were hallucinations and repetitions which result in false-positive messages, none of the actual messages were lost. This is an important result, as the repeated packets and hallucinations can be managed with sequences and padding, lost messages would cause more difficult problems.

7.6.3 Wireless Test

Environment Description

In this section we describe the experiments undertaken to evaluate our BBC implementation in a wireless environment.

A simple form of amplitude-shift keying (ASK) modulation named On-off keying (OOK) is used in order to have OR channel behavior during wireless communications. OOK modulation is a way of representing digital data as the presence or absence of a carrier wave.

RF ASK wireless receiver and transmitter modules operating at 315MHz are used for the wireless tests. Figure 7.9 shows the physical setup for the wireless tests. All transmitters were placed at equal distance to the receiver.

Results

The tests performed for the wireless communication are similar to the wired tests. Once again, all nodes are transmitting asynchronously among them. Although noise
is present during the communications the impact of extra bits in the received packet was also tested. Table 7.4 summarizes the experiments.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>No Noise</td>
<td>all nodes transmitting with no noise</td>
</tr>
<tr>
<td>30% Noise</td>
<td>all nodes transmitting + 30% of random 1 bits in the packet</td>
<td></td>
</tr>
<tr>
<td>50% Noise</td>
<td>all nodes transmitting + 50% of random 1 bits in the packet</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4: Wireless tests description

The hallucination rate and number of repetitions were also recorded for all the wireless tests. For this test, we evaluate from 1 to 3 nodes transmitting with each
node transmitting a total of 800 packets. This corresponds to 20 minutes of continuous transmission. Table 7.5 presents the results.

<table>
<thead>
<tr>
<th>Type</th>
<th>#Nodes</th>
<th>Halluc.(%)</th>
<th>Rep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Noise Wireless</td>
<td>1</td>
<td>0</td>
<td>347</td>
</tr>
<tr>
<td>No Noise Wireless</td>
<td>2</td>
<td>0</td>
<td>464</td>
</tr>
<tr>
<td>No Noise Wireless</td>
<td>3</td>
<td>0</td>
<td>1189</td>
</tr>
<tr>
<td>30% Noise Wireless</td>
<td>2</td>
<td>9.31</td>
<td>281</td>
</tr>
<tr>
<td>50% Noise Wireless</td>
<td>2</td>
<td>9324.56</td>
<td>9651</td>
</tr>
</tbody>
</table>

Table 7.5: Wireless Results

Comments

Once again, for the wireless test, the results were obtained in uncontrolled environment, so external noise introduced extra 1 bits on the final OR channel.

When noise is not present there are no hallucinations. This is due to the fact that there are only up to 3 nodes transmitting and the ambient noise was not strong enough to generate extra 1 bits. But when noise is present (30% and 50%) hallucinations start to happen. However, similar to the wired implementation, the receiver was able to get the messages of all the transmitters without losing any packets.
Chapter 8

BBC as a MAC Layer

This Chapter describes BBC as a deterministic MAC layer for wireless sensors actuators networks. Section 8.1 presents a brief introduction about the importance of implementing BBC as a MAC layer. Section 8.2 describes the hardware and software used for this Chapter. Section 8.3 presents the tests and results used to select the desired BBC parameters. Section 8.4 describes a methodology to select the parameters for BBC if it is desired to used a different hardware platform. Section 8.5 presents the physical setup used to conduct the tests. Finally, Sections 8.6, 8.7 and 8.8 show the transmission space characterization, results and comments, respectively.

8.1 Introduction

Non deterministic delays may play an important role in ad hoc networks.

As mentioned before, a major concern with the protocols described in Section 3 is the management of the shared transmission medium [28, 29]. This issue is also
present during the implementation of RT-MAC in Section 6. During the states CreateNew and JoinGroup from Figure 6.5 the nodes can transmit simultaneously, because they are not yet organized in the TDMA frame or they are transmitting during the dynamic slot time, respectively. In this case the communication will suffer from self jamming as a consequence of multiple nodes attempting to transmit at the same time. The solution to this problem is solved by using the WiFi data link layer which implements a RTS and CTS packets to claim the wireless medium for transmission. However, this solution introduces non-determinism as it is necessary to wait for random delay before trying to transmit again.

In Section 4, it is presented the critical path concept which includes the four main components that contribute to the communication delays (t_{send}, t_{recv}, t_{acc}, and t_{prop}). In existing algorithms for NCS, synchronization and heuristics, the communication delays are handle primarily by methods that estimate and correct these sources of error. A communication model might be desired to be included in the estimation methods.

The rest of the this section describes the implementation, tests and results of BBC as a MAC layer. It will also present the transmission medium characterization that can be used as a communication model for the algorithms that require a communication model.

### 8.2 Hardware Setup

The algorithms and microcontroller hardware used for this section are the same as the ones described in Section 7.6. The only difference relies on adding a “local
behavior” task on the network layer which will allow the implementation of additional algorithms. Figure 8.1 presents the tasks used to implement the tests for this section.

![Diagram of network layers](image)

Figure 8.1: BBC Tasks + Behavior Tasks

### 8.3 BBC Parameters Selection

It is necessary to evaluate the timing boundaries of the tasks related to the BBC in order to understand the results presented later in Section 8.7. The time needed to transmit and receive packets will influence the test timings. This is part of the characterization of the sending and receiving delays ($t_{send}$ and $t_{recv}$) mentioned above in Section 8.1.

The relevant BBC parameters (described in [98, 103]) are the following:

- $c =$ length of a packet (transmission space size)
- $d =$ number of bits per message
The selected values for the parameters presented below were chosen by considering the limitations of the selected technology (available memory and cpu speed). It is important to mention that different values can be selected if the hardware used to implement BBC is different.

The size of $d$ will influence the time that BBCDecoder will take to decode messages. For the tests conducted in this paper, all tasks are running at 100 MHz each and each message will have $d = 24$ bits.

BBCDecoder was left running for 10mins and it was provided different types of messages to decode. Table 8.1 presents the average and standard deviation decoding time of BBCDecoder task from 1 message up to 16 messages sent simultaneously. Figure 8.2 shows the average time, standard deviation and the respective bit time considering $d = 24$. The bit time was calculated by adding to the average time 2 times the standard deviation as a safety factor ($bit_{time} = avg + 2 \times \sigma$).

<table>
<thead>
<tr>
<th># Msg</th>
<th>Average (us)</th>
<th>$\sigma$ (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>162.64</td>
<td>2.04</td>
</tr>
<tr>
<td>2</td>
<td>235.19</td>
<td>3.37</td>
</tr>
<tr>
<td>3</td>
<td>306</td>
<td>8.79</td>
</tr>
<tr>
<td>4</td>
<td>389.06</td>
<td>10.66</td>
</tr>
<tr>
<td>5</td>
<td>455.33</td>
<td>11.88</td>
</tr>
<tr>
<td>6</td>
<td>529.37</td>
<td>9.85</td>
</tr>
<tr>
<td>7</td>
<td>614.56</td>
<td>17.43</td>
</tr>
<tr>
<td>8</td>
<td>685.44</td>
<td>0.38</td>
</tr>
<tr>
<td>9</td>
<td>799.29</td>
<td>18.39</td>
</tr>
<tr>
<td>10</td>
<td>881</td>
<td>21.34</td>
</tr>
<tr>
<td>11</td>
<td>969.75</td>
<td>26.37</td>
</tr>
<tr>
<td>12</td>
<td>1051.8</td>
<td>18.48</td>
</tr>
<tr>
<td>13</td>
<td>1166.6</td>
<td>31.84</td>
</tr>
<tr>
<td>14</td>
<td>1250.3</td>
<td>26.4</td>
</tr>
<tr>
<td>15</td>
<td>1372.3</td>
<td>31.2</td>
</tr>
<tr>
<td>16</td>
<td>1452.63</td>
<td>3.97</td>
</tr>
</tbody>
</table>

Table 8.1: BBCDecoder decoding time (XMOS)
For the remaining tests, it was selected 12 as the maximum number of nodes that can transmit simultaneously. With this, the transmission of one bit can not be less than 1088.75 us (1051.8 us + 2*18.48 us) in order to guarantee the decoding of 12 messages sent simultaneously. The selected bit time was rounded up to 1500 us ($bit_{time} = 1.5ms$) as a safety factor.

The transmission space should not be more than 30% full to avoid hallucinations during the decoding stage [103]. With this, the size of $c$ will be $c = \lceil (d \times 12) / 0.3 \rceil = 960$ (rounded to 1000). The minimum transmission time for each node will be $c \times bit_{time} = 1000 \times 1.5ms = 1.5s$ per message.

Figure 8.3 shows the time to transmit one message versus the number of nodes that can transmit simultaneously (with no more than 30% of data on the transmission space) using the selected technology.
As mentioned above the selected BBC values were chosen by considering the limitations of the selected technology. However, BBC can benefit with the increase of cpu speed. Table 8.2 presents the results for the BBCDecoder task using two other platforms available in the DU2SRI lab. The platforms used were the 2nd (2GHz) and the 6th (2.7Ghz) generation controllers.

Figure 8.4 shows the comparison between the 3 platforms. The BBC parameters used were the same for all platforms.

### 8.4 System Parameterization Methodology

BBC is not restricted to the technology described in this thesis. Below it is presented a methodology that can be used to tune the BBC parameters according to the technology desired to use.

**Requirements**
<table>
<thead>
<tr>
<th># Msg</th>
<th>Average (us)</th>
<th>σ (us)</th>
<th>Average (us)</th>
<th>σ (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.552</td>
<td>6.744</td>
<td>10.321</td>
<td>1.198</td>
</tr>
<tr>
<td>2</td>
<td>49.108</td>
<td>5.995</td>
<td>12.350</td>
<td>1.095</td>
</tr>
<tr>
<td>3</td>
<td>62.724</td>
<td>3.334</td>
<td>16.058</td>
<td>1.191</td>
</tr>
<tr>
<td>4</td>
<td>79.953</td>
<td>6.076</td>
<td>19.210</td>
<td>1.211</td>
</tr>
<tr>
<td>5</td>
<td>94.108</td>
<td>7.530</td>
<td>22.178</td>
<td>1.129</td>
</tr>
<tr>
<td>6</td>
<td>109.380</td>
<td>10.798</td>
<td>25.011</td>
<td>0.772</td>
</tr>
<tr>
<td>7</td>
<td>126.970</td>
<td>8.019</td>
<td>28.477</td>
<td>1.234</td>
</tr>
<tr>
<td>8</td>
<td>140.250</td>
<td>7.240</td>
<td>30.902</td>
<td>0.841</td>
</tr>
<tr>
<td>9</td>
<td>164.360</td>
<td>7.556</td>
<td>35.992</td>
<td>1.258</td>
</tr>
<tr>
<td>10</td>
<td>178.580</td>
<td>7.254</td>
<td>39.014</td>
<td>1.468</td>
</tr>
<tr>
<td>11</td>
<td>198.830</td>
<td>11.404</td>
<td>42.850</td>
<td>1.547</td>
</tr>
<tr>
<td>12</td>
<td>216.050</td>
<td>12.005</td>
<td>45.500</td>
<td>1.284</td>
</tr>
<tr>
<td>13</td>
<td>239.070</td>
<td>14.857</td>
<td>50.945</td>
<td>1.910</td>
</tr>
<tr>
<td>14</td>
<td>255.580</td>
<td>14.110</td>
<td>53.895</td>
<td>1.635</td>
</tr>
<tr>
<td>15</td>
<td>282.850</td>
<td>19.417</td>
<td>59.760</td>
<td>1.842</td>
</tr>
<tr>
<td>16</td>
<td>309.85</td>
<td>19.520</td>
<td>65.265</td>
<td>1.750</td>
</tr>
</tbody>
</table>

Table 8.2: BBCDecoder decoding time (2nd (2GHz) and 6th (2.7GHz) Generation)

Figure 8.4: Time to Transmit 1 Message vs # Nodes Transmitting Simultaneously (Different Platforms)
There are two requirements that can be used to tune BBC:

1. Select the number of nodes that can transmit (N) or

2. Select the desired transmission message rate ($MSG_{TIME}$).

Both requirements are dependent of each other. If one of the above values does not meet the requirements, then it is necessary to change the values until it is achieved a desired and balanced result.

**Find the Bit Time ($bit_{time}$)**

- First, select the desired number of bits to transmit (e.g.: $d = 24$ bits);
- Second, test the BBCDecoder function on the platform desired to use and obtain a table identical to Tables 8.1 and 8.2. The table will have the $bit_{time}$ of the correspondent number of simultaneous decoded messages.

**Get Parameters**

- If the requirement is N then:
  - Calculate the transmission space, $c = \frac{d \times N}{0.3}$, where 0.3 is the safety factor that corresponds to no more than 30% of 1’s in $c$ to avoid hallucinations;
  - Calculate the transmission message rate, $MSG_{TIME} = c \times bit_{time}$, where $bit_{time}$ is obtained from the table calculate in the previous step.

- If the requirement is $MSG_{TIME}$ then:
  - Select a $bit_{time}$ from the table of the previous step and find $c = \frac{MSG_{TIME}}{bit_{time}}$;
  - Find $N = \frac{c \times 0.3}{d}$
8.5 Physical Setup

The physical setup is shown in Figure 8.5. It consists of 5 nodes with one XMOS board and one AM emitter and receiver radio per node (Figure 8.6). There are available two I/O pins on each node that will be raised upon each message arrival or transmission during the synchronization protocols tests.

Those pins are attached to an external logic analyzer that will record the message time reception or transmission.

The logic analyzer used was the OWON MSO8102T with 66MHz of bandwidth and 16 channels. The select sampling rate for all the tests was 20KHz (50us).

Figure 8.5: Physical Setup
8.6 Transmission Medium Characterization

The main idea behind the tests is the following: to have one node (master node) that will transmit packets (beacon packets) with a certain rate. The receivers (slave nodes) will pull the I/O pin high for each received beacon packet. For each pulse transmitted by the master node the difference between the transmitted and the received beacon packet was computed for all the receivers. During all tests none of the pulses were lost.

An internal and external clock was provided to the BBCSampler task to test if the nodes need to be synchronized among them. With the internal clock, all nodes run independently of each other and they have their own clock (Not Sync). With the external clock, the BBCSampler task of all nodes have the same clock (Sync) provided by a function generator.

The number of nodes that are transmitting was also changed to test if simultaneous communication will influence the beacon packet reception time.
Table 8.3 describes the performed tests. For all tests, the master node broadcasts 1000 beacon packets with an interval of 2 seconds. A total of 4000 values were obtained per test.

<table>
<thead>
<tr>
<th></th>
<th>Clock</th>
<th># Nodes Transmitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>Not Sync</td>
<td>1</td>
</tr>
<tr>
<td>Test 2</td>
<td>Not Sync</td>
<td>5</td>
</tr>
<tr>
<td>Test 3</td>
<td>Sync</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 8.3: Tests Description

### 8.7 Results

Table 8.4 shows the results obtained. The average column presents the average of the difference between the transmitted and the received beacon packet for all the receivers with the master. The standard deviation (σ) column presents the dispersion that exists from that average.

<table>
<thead>
<tr>
<th></th>
<th>Average (s)</th>
<th>σ (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>1.4999</td>
<td>430</td>
</tr>
<tr>
<td>Test 2</td>
<td>1.5</td>
<td>427</td>
</tr>
<tr>
<td>Test 3</td>
<td>1.5</td>
<td>430</td>
</tr>
</tbody>
</table>

Table 8.4: Tests Results

The Probability Density Function (PDF) was computed to characterize the transmission medium using BBC. Figures 8.7, 8.9 and 8.11 shows the PDF for the respective test.

The Cumulative Distribution Function (CDF) was also computed for a 95% of confidence bounds to make the standard deviation more informative. Figures 8.8, 8.10 and 8.12 shows the CDF for the respective test.
Figure 8.7: Difference Between Master and Slaves Packet Reception Time vs PDF (Density) - [Test 1] Beacon Time = 2s (Not Sync)

Figure 8.8: Difference Between Master and Slaves Packet Reception Time vs CDF (Cumulative Density) - [Test 1] Beacon Time = 2s (Not Sync)
Figure 8.9: Difference Between Master and Slaves Packet Reception Time vs PDF (Density) - [Test 2] Beacon Time = 2s (Not Sync)

Figure 8.10: Difference Between Master and Slaves Packet Reception Time vs CDF (Cumulative Density) - [Test 2] Beacon Time = 2s (Not Sync)
Figure 8.11: Difference Between Master and Slaves Packet Reception Time vs PDF (Density) - [Test 3] Beacon Time = 2s (Sync)

Figure 8.12: Difference Between Master and Slaves Packet Reception Time vs CDF (Cumulative Density) - [Test 3] Beacon Time = 2s (Sync)
8.8 Comments

By performing the tests described above the transmission medium using BBC as a Medium Access Control (MAC) layer can be characterized.

The results in Table 8.4 shows an average of 1.5 s. This is related with the message transmission time described in Section 8.3. If the message transmission time is subtracted to all the received beacon packets the average will be zero, meaning that on average the nodes have 0 s delay receiving the beacon packets.

A non-parametric probability function was used for the PDF results in order to get an idea of a distribution function that fits the data. For all tests the received packet times have approximately an uniform distribution, meaning that all values are equally probable.

The CDFs results for all tests show that it can be approximated by a linear function.

This can be relevant for the estimation and correction algorithms that rely on a model for the communication environment. As mentioned in Section 3.2, the authors in [43] present a comparison for different packets arrival models (Figure 3.5). The main disadvantage of the models presented in Figure 3.5 is that if the environment changes the model might not be applicable.

On the other end, the linear function obtained with the BBC can be the same even when the environment changes. To adjust it, it is just a matter of changing a couple of parameters on the BBC such as the transmission space size (described in Section 8.3) or the number of nodes that can transmit simultaneously. The behavior will be the same due to the way that BBC works.
Chapter 9

BBC / RT-MAC

This Chapter describes the merge of BBC with RT-MAC. Section 9.1 presents an introduction of BBC and RT-MAC structure. Sections 9.2 and 9.3 describe the data link layer and network layer for BBC with RT-MAC, respectively. Section 9.4 presents time line diagrams examples. Finally, Section 9.5 shows the hardware implementation.

9.1 Introduction

In Section 6, RT-MAC was implemented using the MAC layer of WiFi as the transmission medium manager. As explained in Section 8.1, the WiFi data link layer uses the RTS and CTS technique to claim the wireless medium for transmission. This solution introduces non-determinism as it is necessary to wait for random delay before trying to transmit again.

This Section presents the merge of BBC with RT-MAC (Figure 9.1). The main goal is to use BBC as the data link layer and then implement RT-MAC on top of
it (network layer). The formation network techniques described in Section 3.5 can then be implemented in the application layer and if it is required a model of the transmission medium, it can be obtained from Section 8.6.

Figure 9.1: RT-MAC place in the OSI model with BBC as under layer

The Sections below describe each layer in more detail.

9.2 Data Link Layer

The data link layer contains the BBC MAC layer described in Section 7.6. It has three tasks running at the same time: BBCEncoder, BBCDecoder and BBCSampler. All the communications between tasks are done through XMOS channels.
9.3 Network Layer

The network layer contains a modified version of the RT-MAC described in Section 6. The main reason is because RT-MAC needs to be adapted to the layer that is below. In Section 6, the transmission medium was managed by the WiFi data link layer and RT-MAC needed to accommodate the non determinism due to the random access to transmission medium. With BBC in the data link layer, multiple and simultaneous transmission are possible.

Only the Sections where RT-MAC is different are described below. The other Sections remain identical to the original RT-MAC described in Section 6.2.

9.3.1 Frame Description

The RT-MAC frame structure to use with BBC is the same basic RT-MAC frame structure from Section 6.2. It has the same parameters, Data Transmission Time (DTT), Data Transmission Size (DTS), Maximum Transmission Rate (MTR), Frame Length (FL) and Guard Period (GP).

In this case it is not necessary to include the Packet Repetition (PR) parameter. BBC allows multiple and simultaneous transmissions, so there is no interference due to packets collision.

The equation to get the number of nodes allowed in a RT-MAC frame is the following:

\[ N = \left\lfloor \frac{FL}{(GP + DTT)} \right\rfloor \]  

(9.1)

Where \( DTT = \frac{DTS}{MTR} \).
9.3.2 Scheduling

The scheduling in this case is easier than the one described in Section 6.2. The original RT-MAC is divided into static and dynamic slots. This was the solution created to allow nodes to transmit without competing for the transmission medium and to avoid packets collision. It is not necessary to have static or dynamic slots due to the ability of simultaneous transmissions with BBC. Nodes can transmit whenever they want that the receivers are able to decode the messages separately.

However, in the Section below the concept of static slots will be maintained to represent the organized slots present in RT-MAC.

9.4 Time-Line Examples

This section presents the time line diagrams for different examples of BBC with RT-MAC.

Figure 9.2 shows the basic time line diagram.

![Figure 9.2: BBC with RT-MAC Scheduler - Basic Example](image.png)
Slot color description:

- Green: leader slot;
- Dark blue: normal slot;
- Light blue: available slot;
- Dark orange: normal slot;

Note: Dark orange is not shown in Figure 6.12. It will be used later for joining and leaving formation examples.

9.4.1 Create Group

Figure 9.3 represents the timeline to create a group of robots. There is a period where all robots broadcast the information and build the table (e.g., organized by unique identifier) according to their surroundings. After that period, they start transmitting the information according to RT-MAC communication scheme.

Figure 9.3: BBC with RT-MAC Scheduler - Create a Group
9.4.2 Lost Leader / Assign New Leader

Figure 9.4 represents a assignment of a new leader.

![Diagram showing Lost Leader / Assign New Leader](image)

Figure 9.4: BBC with RT-MAC Scheduler - Lost Leader / Assign New Leader

9.4.3 Join Group

For the join group behavior, it is presented 3 time line diagrams.

The first time line diagram (Figure 9.5) shows how one robot can join the formation. In this case robot 3 enters in a scheduler learning phase. Then it tries to join the group by transmitting whenever it wants. Then it gets a response from the leader and successfully joins the group.

![Diagram showing One Robot Join Example](image)

Figure 9.5: BBC with RT-MAC Scheduler - One Robot Join Example
The second time line diagram (Figure 9.6) shows how three robots can join the formation. In this example all robots first learn the scheduler by listening the environment. Robot 3, 4 and 5 successfully transmitted the request to join simultaneously. In this case the leader accepts each robot at the time but it could accept all at once.

In this example the leader accepts each robot at the time to illustrate the transmission hierarchy on the static slots changing while the robots are joining the group.

Figure 9.6: BBC with RT-MAC Scheduler - Three Robots Join Example

The third time line diagram (Figure 9.7) shows a robot join failure. All the static slots are taken by 8 robots. Robot 9 attempts to joint the group but the leader refuses the request.

9.4.4 Leave Group

The time line diagram in Figure 9.8 represents one robot leaving formation. Robot 3 sends a message communicating that it is leaving. The leader answers and the robot leaves the group.
Figure 9.7: BBC with RT-MAC Scheduler - Robot Join Failed Example

Figure 9.8: BBC with RT-MAC Scheduler - One Robot Leave Example
9.5 Hardware Implementation

The hardware used to implement RT-MAC with BBC is the same hardware from Section 8.5. Each node has one XMOS board and one AM emitter and receiver radio per node.

RT-MAC with BBC was tested up to 5 modules (Figure 9.9).

Figure 9.9: BBC with RT-MAC - Physical Setup
Chapter 10

Conclusions and Future Work

10.1 Conclusion

This research presents three components which aim to reduce the communication delay variability for a group of robots. These components are: a new methodology (benchmark) that evaluates the I/O responsiveness of microprocessors; an event-based communication protocol, in which nodes transmit in a TDMA fashion; and a novel communication scheme that enables deterministic and simultaneous communications.

Regarding the technology selected, a novel and nonexistent I/O benchmark was applied to 3 different architectures. The results show that XMOS exhibited better characteristics in terms of response latency, in particular, the worst case latency and the jitter being significantly less than the other architectures. The event-based communication protocol (RT-MAC) enables the transmission of packets in a TDMA fashion, without the need of global clock synchronization techniques. It incorporates the advantages of several different types of TDMA protocols into this novel
event-based communication protocol. The novelties are the following: easy tuning of RT-MAC protocol parameters; and combination of static slots with dynamic slots, which enables adding and removing nodes dynamically in the group. The novel communication scheme (BBC) enables deterministic communications by allowing senders to transmit without regard for the state of the medium or coordination with other senders, and receivers can tease apart messages sent simultaneously with high probability of success. This new communication scheme was never implemented as a MAC layer or even adapted for embedded systems. Finally all the three components were merged in the following manner: BBC was implemented on the data link layer as a MAC protocol and RT-MAC was implemented on the network layer of the OSI model. Both, BBC and RT-MAC, were implemented on XMOS that guarantees deterministic real-time tasks.

The techniques used to ensure the shape formation of a group of robots (e.g.: behavior-based control or graph theory) can be implemented on the application layer of the OSI model. The communications are handled by the underneath layers (RT-MAC / BBC) that guarantee order and deterministic communications. It is also provided a transmission medium model of BBC in case the selected formation technique requires a model of the communication environment.

10.2 Future Work

The research performed up to this point can be advanced along two paths: hardware and algorithm development. The hardware development throughout this research is suitable for demonstrating certain fundamental concepts. Therefore, the future work must involve the implementation of the communication scheme (BBC) in
a dedicated Application-Specific Integrated Circuit (ASIC) to improve for example the decoding speed stage. Moreover, regardless the good results obtained for the low values of hallucinations present in BBC, it will be benefit to research techniques or methods to complete eliminate them.

This research explored the communication side present in shape formation algorithms. In the immediate future, the integration of these algorithms (e.g.: virtual structures or behavior-based) with the proposed architecture can be explored and incorporated with the control algorithms.

Moreover, any application that requires coordination or shape formation among a group of robots (e.g.: convoy protection or detection and monitoring of oil spills in water), will benefit by implementing or use the proposed architecture as the communication medium.

Finally, Wireless Sensor Networks (WSNs) are networks in which nodes are low-cost sensors that can communicate with each other in a wireless manner, have limited computing capability, memory and operate with limited battery power. Communications can be the task that consumes the majority of the available energy. Energy conservation is one of the most important aspects to be considered in these networks. The use of BBC as the communication scheme will help saving some of this available energy.

10.3 Publications

First Author


137


To Publish


Second Author


138

Bibliography


