Bandwidth-Aware Distributed Ad-hoc Grids in Deployed Wireless Sensor Networks

Elisa Rondini

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University College London.

Department of Computer Science

University College London

2010

To Ettore
my best travel companion

To Ivanna and Franco
for having taught me life
and being always there

I, Elisa Rondini, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis. Financial support for this work, as part of the Divergent Grid project, has been provided by EPSRC under grant number EP/C534891.

Abstract

Nowadays, cost effective sensor networks can be deployed as a result of a plethora of recent engineering advances in wireless technology, storage miniaturisation, consolidated microprocessor design, and sensing technologies.

Whilst sensor systems are becoming relatively cheap to deploy, two issues arise in their typical realisations: (i) the types of low-cost sensors often employed are capable of limited resolution and tend to produce noisy data; (ii) network bandwidths are relatively low and the energetic costs of using the radio to communicate are relatively high. To reduce the transmission of unnecessary data, there is a strong argument for performing local computation. However, this can require greater computational capacity than is available on a single low-power processor. Traditionally, such a problem has been addressed by using load balancing: fragmenting processes into tasks and distributing them amongst the least loaded nodes. However, the act of distributing tasks, and any subsequent communication between them, imposes a geographically defined load on the network. Because of the shared broadcast nature of the radio channels and MAC layers in common use, any communication within an area will be slowed by additional traffic, delaying the computation and reporting that relied on the availability of the network.

In this dissertation, we explore the tradeoff between the distribution of computation, needed to enhance the computational abilities of networks of resource-constrained nodes, and the creation of network traffic that results from that distribution. We devise an application-independent distribution paradigm and a set of load distribution algorithms to allow computationally intensive applications to be collaboratively computed on resource-constrained devices. Then, we empirically investigate the effects of network traffic information on the distribution performance. We thus devise bandwidth-aware task offload mechanisms that, combining both nodes computational capabilities and local network conditions, investigate the impacts of making informed offload decisions on system performance.

The highly deployment-specific nature of radio communication means that simulations that are capable of producing validated, high-quality, results are extremely hard to construct. Consequently, to produce meaningful results, our experiments have used empirical analysis based on a network of motes located at UCL, running a variety of I/O-bound, CPU-bound and mixed tasks. Using this setup, we have established that even relatively simple load sharing algorithms can improve performance over a range of different artificially generated scenarios, with more or less timely contextual information. In addition, we have taken a realistic application, based on location estimation, and implemented that across the same network with results that support the conclusions drawn from the artificially generated traffic.

Acknowledgements

The PhD has been one of the most challenging and interesting experiences I had in my life. It has been a constant process of learning and discussing yourself, your work and other people's work. It changes you as a person and makes you grow, stronger.

I would like to thank Stephen Hailes and Wolfgang Emmerich whom I am proud to call my supervisors. They thoughtfully guided my research from nonsense to sense by constantly pushing me into clarifying the purpose and execution of my research ideas. Their guidance, experience and encouragement have been invaluable for my research. Thanks again to Steve who allowed me to disappear for a couple of months to take up an internship at NEC (Heidelberg, Germany).

A special thanks is dedicated to Ettore, without whom my life would be meaningless, and to whom this dissertation is dedicated. Thanks for being always there, for supporting, helping me and for constantly reminding me how ones personal life should always come before ones career.

Special thanks to everyone that supported and helped me during my research in London, Oxford, Cambridge and Ottawa (Canada). On the UCL side, a special thanks goes to Licia Capra (and Luca) for the valuable research discussions and for being really a wonderful friend, always there for a chat or for advice. Again, thanks very much indeed to the students, staff and academics of the Department of Computer Science at UCL, with special attention to the members of the Mobile Systems Group, Networks Research Group and Software Systems Engineering Group. On the Birkbeck side, thanks to George Roussos for the valuable research discussions at the beginning of my PhD carrier. On the Oxford side, a special thanks goes to Niki Trigoni, Alexandre Guitton and Antonios Skordylis for the friendly chats and PhD advices. On the Cambridge side, thanks to Cecilia Mascolo for being so supportive during every PhD step. A special thanks to Li Li for her incredible, transoceanic collaboration and support both as a researcher and as a friend.

From a more personal perspective, thanks to my parents, Ivanna e Franco who always believed in me since the beginning, and supported me in every possible way. Thanks also to Ettore's family: Claudia, Andrea, Elena and Riccardo. A special thanks to my uncles Giorgio, Gianni, aunts Fausta and Luisa and cousins Cinzia, Luca and Stefano, to name a few.

What is life without friends? I have been really lucky from this point of view, because I could not have met better ones here in London. I really need to start from Chris, my best friend. Thanks for each single moment spent together, for the long chats, for the wonderful and unforgettable experiences in and out UCL. Thanks for our special, true and so unique friendship. Even if everything had gone bad during

my stay in England, his great friendship would have still made it worth coming here. And, of course, a special thanks goes to Jasmin, Beppe and Claudio, who were always so supportive and motivating.

Thanks to all the friends I made here, Adam, Afra, Aitor, Alan, Anastasia, Anders, Andrea, Andy, Bence, Beppe, Bozena, Bruno, Claudio, Clovis, Costin, Cristina, Daniele (Borsaro), Daniele (Quercia), Daria, Dimitris, Endri, Enrico, Felipe (with special thanks to him and Sabine, also for the wonderful Heidelberg experience), Franco, Genaina, Ilias, James, Jidtima, Jo, Leticia, Liam, Manish, Matteo, Max, Maxamed (Mo), Michele, Mirco, Panteha, Pan Xueni, Rae, Salvo, Selma, Socrates, Sonia (together with Francois and Lisie), Stefania, Stefano, Torsten, Valentina, Vito and Vladimir. I could really write a whole book only about them, and how great they all are, and it would be much longer than this thesis (and I am glad about that). Thanks again to all my friend in Novellara (Silvia, Alessia, Elena, Chiara, Sara and guys) and Correggio (Rita, Giulia, Ilaria, Linda and guys), that were always there to listen to my crisis.

A special thanks is addressed to Google that twice awarded me during the PhD with the Google Anita Borg Scholarship, both with the role of Finalist and that of Scholar. With the awareness of belonging to a very special group of women that decided to defy prejudices by working in Computer Science, I greatly thank Prof. Tracy Camp and Prof. Wendi Heinzelman that awarded me with two fellowships to organise and lead Networking Networking Women (N^2Women) events at two major conferences in Computer Science, namely ACM SenSys 2008 and IEEE MILCOM 2008. Such experiences have been truly valuable and inspiring for me: they helped me understanding the crucial importance for women in Computer Science to get together, know each other and share each other experiences while, at the same time, they pushed me in actively promoting and building such human and research networks of connections.

Finally, I would like to thank the anonymous reviewers of the papers I have published, who helped me improving the work, the people I met at the conferences I attended and the EPSRC financial support without which this thesis would not have been written.

Contents

1 Introduction **1**

1.1 Background . . . 1

1.2 Problem Statement . . . 4

1.3 Research Hypothesis . . . 5

1.4 System Assumptions . . . 6

1.5 Experimental Assumptions . . . 6

1.6 Contributions . . . 8

1.6.1 Distributed Wireless Ad-hoc Grid Paradigm . . . 8

1.6.2 Network Conditions Impact on Distribution in Deployed Systems . . . 8

1.6.3 Impact of Heterogeneous Applications Distribution . . . 9

1.6.4 Localisation Case Study . . . 9

1.7 Publications . . . 9

1.8 Thesis Outline . . . 10

2 Requirements and Background **11**

2.1 Scenario Requirements and Background . . . 11

2.2 Related Work . . . 12

2.2.1 Distributed Processing in Wireless Sensor Networks . . . 12

2.2.2 Network Communication Issues in WSNs . . . 18

2.2.3 Operating Systems for WSNs . . . 22

2.3 Discussion . . . 28

3 Distributed Wireless Ad-hoc Grids **29**

3.1 Distributed Wireless Ad-hoc Grids Paradigm . . . 29

3.2 Application Requirements for DWAG Applicability . . . 32

3.3 DWAG System Flow Description . . . 33

3.4 Related Work . . . 35

3.4.1 Grid Computing and DWAG Features . . . 35

3.4.2 DWAG vs. Sensor Grid Systems . . . 37

3.5 Discussion . . . 38

4 Impact of Network Conditions on DWAG in Deployed Systems **39**

4.1 Network Contention Impact on Distribution . . . 40
4.2 Load Distribution Algorithms . . . 41
4.2.1 The Auction Algorithm . . . 43
4.2.2 The Lookup List Algorithm . . . 45
4.3 Bandwidth-Aware Task Scheduling Heuristic . . . 49
4.4 CPU and Bandwidth Load Computation . . . 50
4.4.1 CPU Load Computation . . . 51
4.4.2 Bandwidth Load Computation . . . 52
4.5 Related Work . . . 55
4.5.1 Load Distribution Principles . . . 55
4.6 DWAG Evaluation with Bandwidth-Aware Task Scheduling Heuristic . . . 57
4.6.1 Experimental Setup . . . 57
4.6.2 UCL-CS HEN TMote Sky Sensor Testbed . . . 62
4.6.3 Experimental Results with Single Grid Activator . . . 64
4.6.4 Experimental Results with Multiple Grid Activator . . . 69
4.7 Summary of Results . . . 73
4.8 Discussion . . . 74

5 Impact on Distributing Heterogeneous Tasks **75**

5.1 Heterogeneous Application Types . . . 75
5.2 Lower-bound Computation . . . 78
5.3 Heuristic Set . . . 84
5.4 Experimental Evaluation . . . 86
5.4.1 1^{st} Experimental Setup and Results (Auction vs. Lookup List) . . . 90
5.4.2 2^{nd} Experimental Setup and Results (I/O-Bound Jobs) . . . 91
5.4.3 3^{rd} Experimental Setup and Results (CPU-Bound Jobs) . . . 98
5.4.4 4^{th} Experimental Setup and Results (Balanced Jobs) . . . 105
5.4.5 5^{th} Experimental Setup and Results (Mixed Jobs) . . . 113
5.5 Summary of Results . . . 117
5.6 Discussion . . . 118

6 Localisation Case Study **119**

6.1 Driving Motivation . . . 119
6.2 SASNet Localisation Algorithm Overview . . . 120
6.2.1 Curvilinear Component Analysis . . . 121
6.2.2 CCA Algorithm . . . 122
6.2.3 CCA-MAP Algorithm . . . 123

6.3 DWAG/BATS and CCA-MAP Integration 127
6.3.1 Localisation Tasks Identification 127
6.3.2 Moving from Theory to Practice 129
6.4 CCA-MAP Evaluation with DWAG/BATS 132
6.4.1 Experimental Setup 132
6.4.2 Experimental Results 136
6.5 Summary of Results 137
6.6 Discussion 138

7 Conclusions and Future Work **139**
7.1 Summary of Contributions 139
7.2 Requirements Revision 140
7.3 Critical Evaluation 141
7.4 Future Work 143

List of Figures

2.1 TMote Sky platform. . . . 13
2.2 Hidden terminal problem in wireless networks. . . . 19
2.3 Exposed terminal problem in wireless networks. . . . 20
2.4 The communication primitives in the Rime stack and how they are layered. . . . 25
2.5 The Chameleon architecture. . . . 26

3.1 Distributed Wireless Ad-hoc Grids paradigm. . . . 30
3.2 System flow performed through the DWAG paradigm. . . . 33

4.1 Network contention impact on distribution through DWAG. . . . 40
4.2 BATS computational load definition and computation. . . . 51
4.3 Clear Channel Assessment value probed from Chipcon CC2420 RF transceiver. . . . 53
4.4 BATS bandwidth load computation with overestimated time window. . . . 53
4.5 BATS bandwidth load computation with underestimated time window. . . . 54
4.6 Homogeneous job type description. . . . 58
4.7 RF channel spectrum of IEEE 802.15.4/ZigBee against IEEE 802.11b/WiFi. . . . 59
4.8 Introduction of interferer node to handle heterogeneous network traffic contention. . . . 60
4.9 View of the UCL-CS HEN TMote Sky sensor testbed. . . . 63
4.10 Partial map of the UCL-CS HEN device deployment. . . . 64
4.11 Map of UCL-CS HEN device deployment for experiments with single GA. . . . 65
4.12 Effects of varying TE number with single GA experimentation. . . . 65
4.13 Effects of varying task offload size with single GA experimentation. . . . 67
4.14 Effects of varying task number with single GA experimentation. . . . 68
4.15 Map of UCL-CS HEN device deployment for Experiment A with multiple GA. . . . 69
4.16 Effects of varying task number in Experiment A and multiple GA. . . . 70
4.17 Map of UCL-CS HEN device deployment for Experiment B with multiple GA. . . . 71
4.18 Effects of varying task number in Experiment B and multiple GA. . . . 71
4.19 Auction and Lookup List comparison in Experiments A and B. . . . 72

5.1 Different application types (I/O-, CPU-bound and balanced jobs). . . . 76
5.2 Experiment to compute task duration on a congested or uncongested TE. . . . 79
5.3 TE_{Best} selection within the theoretical lower-bound algorithm. . . . 80

5.4 Task estimated completion time computation within the theoretical lower-bound algorithm. 81
5.5 Map of UCL-CS HEN device deployment for job heterogeneity experiments. 87
5.6 Performance comparison for Auction and Lookup List with I/O-bound jobs. 89
5.7 Performance comparison for Auction and Lookup List with balanced jobs. 90
5.8 Effects of varying TE and $Task_{ID}$ on task completion time in I/O-bound jobs. 92
5.9 Effects of varying TE and $Task_{ID}$ on task duration in I/O-bound jobs. 93
5.10 Effects of varying TE on job duration and task duration in I/O-bound jobs. 93
5.11 Effects of varying GA and $Task_{ID}$ on task completion time in I/O-bound jobs. 96
5.12 Effects of varying GA and $Task_{ID}$ on task duration in I/O-bound jobs. 97
5.13 Effects of varying GA on job duration and task duration in I/O-bound jobs. 98
5.14 Effects of varying TE and $Task_{ID}$ on task completion time in CPU-bound jobs. 100
5.15 Effects of varying TE and $Task_{ID}$ on task duration in CPU-bound jobs. 101
5.16 Effects of varying TE on job duration and task duration in CPU-bound jobs. 101
5.17 Effects of varying GA and $Task_{ID}$ on task completion time in CPU-bound jobs. 103
5.18 Effects of varying GA and $Task_{ID}$ on task duration in CPU-bound jobs. 104
5.19 Effects of varying GA on job duration and task duration in CPU-bound jobs. 105
5.20 Effects of varying TE and $Task_{ID}$ on task completion time in balanced jobs. 107
5.21 Effects of varying TE and $Task_{ID}$ on task duration in balanced jobs. 108
5.22 Effects of varying TE on job duration and task duration in balanced jobs. 108
5.23 Effects of varying GA and $Task_{ID}$ on task completion time in balanced jobs. 110
5.24 Effects of varying GA and $Task_{ID}$ on task duration in balanced jobs. 111
5.25 Effects of varying GA on job duration and task duration in balanced jobs. 112
5.26 Effects of mixed job types on task completion time and task duration (3GAs). 114
5.27 Effects of mixed job types on job duration and task duration (3GAs). 115
5.28 Effects of mixed job types on task completion time and task duration (6GAs). 116
5.29 Effects of mixed job types on job duration and task duration (6GAs). 117

6.1 CCA-MAP localisation algorithm global map patching procedure. 126
6.2 Tasks identification within the CCA-MAP localisation algorithm. 128
6.3 Effects of varying TE on job duration in network with 6 GAs with RF=-22 dBm (A) and 0 dBm (B). 134
6.4 Effects of varying TE on job duration in network with 10 GAs with RF=-22 dBm (A) and 0 dBm (B). 135

List of Tables

3.1 Grid computing vs. cluster computing systems. 36

5.1 Experimental setup for I/O-bound job experimentation. 91
5.2 Anova analysis for job duration varying TE in I/O-bound jobs (Figure 5.10A). 94
5.3 Anova analysis for task duration varying TE in I/O-bound jobs (Figure 5.10B). 94
5.4 Anova analysis for job duration varying GA in I/O-bound jobs (Figure 5.13A). 95
5.5 Anova analysis for task duration varying GA in I/O-bound jobs (Figure 5.13B). 95
5.6 Experimental setup for CPU-bound job experimentation. 99
5.7 Anova analysis for job duration varying TE in CPU-bound jobs (Figure 5.16A). 102
5.8 Anova analysis for task duration varying TE in CPU-bound jobs (Figure 5.16B). 102
5.9 Anova analysis for job duration varying GA in CPU-bound jobs (Figure 5.19A). 105
5.10 Anova analysis for task duration varying GA in CPU-bound jobs (Figure 5.19B). 105
5.11 Experimental setup for balanced job experimentation. 106
5.12 Anova analysis for job duration varying TE in balanced jobs (Figure 5.22A). 109
5.13 Anova analysis for task duration varying TE in balanced jobs (Figure 5.22B). 109
5.14 Anova analysis for job duration varying GA in balanced jobs (Figure 5.25A). 112
5.15 Anova analysis for task duration varying GA in balanced jobs (Figure 5.25B). 112
5.16 Anova analysis for job duration varying job types (Figure 5.27A). 113
5.17 Anova analysis for task duration varying job types (Figure 5.27B). 113
5.18 Anova analysis for job duration varying job types (Figure 5.29A). 115
5.19 Anova analysis for task duration varying job types (Figure 5.29B). 115

6.1 Anova analysis for job duration varying TE with RF=-22 dBm (Figure 6.3A). 134
6.2 Anova analysis for job duration varying TE with RF=0 dBm (Figure 6.3B). 134
6.3 Anova analysis for job duration varying TE with RF=-22 dBm (Figure 6.4A). 135
6.4 Anova analysis for job duration varying TE with RF=0 dBm (Figure 6.4B). 135

Chapter 1

Introduction

1.1 Background

Years of steady advancements in Micro-Electro-Mechanical Systems (MEMS) have seen them gain wide popularity, especially because of the inexpensive production and pervasive deployment of miniaturised nodes with communication, computation, storage, and sensing capabilities. Gumstix [Gumstix, 2010], SunSPOT [SunSPOT, 2010], Arduino [Arduino, 2010], MICAz [MICAz, 2010], Imote2 [Imote2, 2010], TMote Sky [TELOSB, 2010] are just a few examples of the plethora of embedded devices designed to perform one or more dedicated functions, often with real-time computing constraints.

In particular, recent advances in integrated circuitry and wireless communication, fabrication cost reductions, and innovative high-value applications have solidified the usage of Wireless Sensor Networks (WSNs) as a significant component of pervasive computing. So far, WSNs, comprised of small, autonomous, and highly-constrained sensor nodes, have played a crucial role in filling the void between computers and the physical world by enabling continuous, non-intrusive observations of real-world phenomena, often achieving goals that have been unattainable in the past. In particular, WSNs have been usually employed in a wide spectrum of applications ranging from environmental monitoring [Mainwaring et al., 2002], to wildlife tracking [Juang et al., 2002], supervised agriculture [Burrell et al., 2004], home automation [Petriu et al., 2000], building monitoring and control [Deshpande et al., 2005], and emergency response [Costa et al., 2007], to name but a few. Furthermore, recent surveys on WSNs applications [Yicka et al., 2008, Arampatzis et al., 2005, l. F. Akyildiz et al., 2002] have emphasised their use in any kind of context.

However, since the computational capabilities of such devices are growing quickly, while their size keeps shrinking, we believe that, nowadays, WSNs have become powerful enough to play not only a *passive*, but also an *active* role. To date, WSNs have represented a viable solution to cover the former situation, thus to gather information from the environment and make it available to scientists [l. F. Akyildiz et al., 2002], or to use sensed data to decide on actions to take on the environment [l. F. Akyildiz and Kasimoglu, 2004]. On the contrary, we envision their active usage as entities that, because of their computational capabilities, could be able to collaboratively perform more complex and intensive applications. Moreover, since sensor nodes are not only limited in size but because they are also extremely cheap, they can easily and widely be deployed, thus allowing their pervasive use in

our everyday life.

Given the nature of WSNs technology, it is envisioned that there will be, in the near future, a growing interest in instrumenting public spaces with WSNs [Campbell et al., 2006, Murty et al., 2008], thus exploiting their pervasive deployment and use within a variety of fields. Let us now provide an overview of some application scenarios.

Scenario 1 - Emergency

Consider a Chemical, Biological, Radiological or Explosive (CBRE) emergency event occurring in an indoor environment (e.g. buildings, train stations, underground, etc.). If such threats are realised, severe damage may be caused within a relatively limited area. This limited scope means that it is possible, and clearly desirable, for the civil defence forces to respond as quickly and as effectively as possible to ameliorate and contain the effects of the resulting hazards. However, particularly in indoor environments where the effects of such attacks are magnified by containment, rescue services arrive from outside without a detailed picture of what is happening inside and must pause in order to obtain that picture before being able to respond effectively. WSNs could therefore play a crucial role in supporting first responders to obtain such contextual snapshot. For example, before taking decisions about how to act effectively, first responders would need to know the following environmental information: (i) the epicentre of the threat (e.g. fire location); (ii) the spread of a possible gas cloud, fire, smoke; (iii) the level of hazard lethality when entering the area. Thus, for example, the epicentre and spread of a gas cloud could be assessed and, using a model of lethality, the first responders could directly go to the places in which injured people might be located. WSNs would need to provide a small amount of information that, once *locally* gathered and aggregated inside the network, could be externally communicated to the *Command & Control* (C^2) authority in order to inform reactive countermeasures.

Similarly, once first responders have entered the hazardous environment, the external C^2 would constantly need to know the position of personnel, while it moves. Since in indoor environments Global Positioning System (GPS) support cannot be assumed, smart localisation algorithms would thus need to be computed *locally* and a map of nodes delivered to the outside C^2. Simultaneously, aggregated data, gathered from embedded accelerometers and gyroscopes, could draw a real-time history of first responders' movements, identifying if they were standing, walking, or running at each instant of time.

If the scenario represents a situation that is too dangerous to send human responders inside, an alternative approach could be the deployment of a group of autonomous robots, aiming both to explore and look for hazards or victims, without risking human lives. Moreover, the robots could also deploy a network of sensors during the exploration process [Ferranti et al., 2009]. Such a network could be used by humans to monitor how the emergency situation evolves over time (e.g. moving fires, collapsing corridors). Furthermore, paths could be discovered and maintained by the robots from points of interest in the area (e.g. places where victims or hazards are located) to exit points [Ferranti et al., 2008]. In order to accomplish all these tasks, robots need a great amount of computational power, which could be provided by the *locally* and dynamically built sensor network.

These are only few examples of the pervasive deployment of WSNs to assist personnel in handling

emergencies. To date, a considerable amount of work for emergency scenarios has been performed by Professor Galea's Fire Safety Engineering Group (FSEG) within the University of Greenwich. FSEG is, in fact, one of the largest research groups in the world dedicated to the development and application of mathematical modelling tools (i.e. EXODUS [EXODUS, 2010, Galea et al., 1993], SMARTFIRE [SMARTFIRE, 2010, Taylor et al., 1996], AASK database [AASK, 2010, Owen et al., 1998]) suitable for the simulation of fire/evacuation-related phenomena. The group specialises in computational fire engineering, including fire and evacuation modelling and non-emergency pedestrian dynamics. FSEG expertise and modelling tools are used to help solve fire safety, security and pedestrian dynamics problems. MESA [MESA, 2010], SAFECOM [SAFECOM, 2010], RUNES [Costa et al., 2007], U-2010 [U-2010, 2010] and others [Varakliotis et al., 2009] are additional examples of practical scenarios adopting the pervasive deployment and usage of WSNs within emergencies. In particular, U-2010's objective is to provide the most capable means of communication and the most effective access to information to everybody required to act in case of accident, incident, catastrophe or crisis, while using existing or future telecommunication infrastructures.

Scenario 2 - Tracking and surveillance

Other scenarios benefitting from the pervasive deployment of WSNs are those in which devices are adopted for target tracking and localisation [Bhatti and Xu, 2009, Amundson and Koutsoukos, 2009], search and rescue [van Willigen et al., 2009, Ergen et al., 2009] and surveillance operations [Ali et al., 2008].

For example, consider an outdoor environment (e.g. industrial, financial and urban areas, etc.) required to be constantly patrolled for security reasons. Within industrial areas it might be necessary to promptly detect and notify the presence of unauthorised people in order to limit their access to restricted areas. Similarly, within urban areas it might be constantly required to patrol city roads to detect suspicious activities, thus supporting local police in their surveillance activity. Moreover, especially within highly populated areas, it may be required to constantly monitor the level of air pollution. In the presence of massive environmental disasters (e.g. earthquakes, typhoons, fires, flooding, etc.), it might be necessary to rely upon the aid of additional support entities assisting search and rescue operations, disaster recovery and assessing the level of damage.

Whenever such situations occur, swarms of Unmanned Aerial Vehicles (UAVs) could therefore be deployed in the environment to support human personnel in its activities of surveillance, patrolling, tracking, monitoring, search and rescue operations. The SUAAVE project [SUAAVE, 2010] has been recently initiated to deal with such kind of situations. In SUAAVE, swarms of UAVs represent clouds of networked resource-constrained vehicles acting as sensor platforms targeted towards achieving a global objective in an efficient manner. Such flocks of helicopters have the intrinsic property of being formed by individually autonomous vehicles (i.e. not under the direct real-time control of a human) that are however collaboratively able to self-organise to perform the following actions: (i) sense the environment in the most efficient possible way; (ii) respond to node failures; and (iii) report their findings to a base station on the ground. In effect there are three separate capabilities for use in addressing application-specific problems: (i) ground sensing of various types; (ii) atmospheric sampling; and (iii) the ability to bridge

communications, all within a rapidly deployable, survivable, hands-off package. An increasing interest towards the exploration of decentralised, autonomous flocking schemes has been recently detected, thus leading to the creation of several further research projects [Xiong et al., 2010].

Scenario 3 - Transportation Systems

WSNs have also become an important part of real-time sensing, monitoring, and control of critical transport systems [Al-Ars and Kootkar, 2007]. National railway authorities have been recently attracted towards WSN deployments, against wired ones, because of their (i) constrained power consumption, (ii) automatic and dynamic configuration and routing (i.e. mesh networking multi-hop technologies), (iii) reliable self-healing characteristics for long-term installations, (iv) fast adaptation to mobility and environmental changes.

The integration of WSNs within transport systems would thus lead to exploit intelligent trains and railway infrastructures (i) avoiding expensive wire replacements, while (ii) improving system security, remote monitoring and railway safety. Intelligent infrastructures could thus be adopted, from an energy perspective, to assist transport systems in the network self-maintenance and sustainability, while at the same time representing a flexible pollution monitoring system over large urban areas [Gil-Castieira et al., 2008]. Within underground transport systems, WSNs could also be in charge of monitoring environmental conditions, with respect to human movement and autonomous device localisation.

The above examples represent just a few of possible application scenarios, developed recently, that take advantage of WSNs. However, despite their individual characterisations, within each scenario it is possible to identify a set of common requirements. Firstly, each scenario has a set of computationally intensive applications for which distributed processing capabilities are required, since each application may not run easily on individual, portable devices. In fact, even if it were possible to over-engineer the computational capacity of nodes and deploy them on a large-scale, constraints of cost, size and energy consumption may make this impracticable for deployed systems. Secondly, the above mentioned applications are likely to run within environments with heterogeneous network conditions. Since in none of the above scenarios can wired communication be assumed (e.g. it may be interrupted because of disruption to buildings in emergencies, it may not be an option within UAVs scenarios in which swarms of helicopters exclusively communicate with each other wirelessly, etc.), devices are expected to dynamically and *locally* collaborate with each other to achieve a common goal, communicating through wireless radio communications even under heterogeneous and heavy network conditions. In such scenarios, the main aim is therefore to improve the overall computing capacity of a system by making intelligent offload decisions under severe environmental network conditions.

1.2 Problem Statement

The real deployment of scenarios like those described above requires that we address several challenges that have not been tacked in the literature thus far.

First Requirement: Impact of Distribution

Although the computational capabilities of WNSs devices are growing quickly, their CPUs, memory size, as well as network bandwidth and latency, will continue to lag several orders of magnitude behind their fixed counterparts. Thus single devices, alone, might not able to perform computationally intensive applications. On the other hand a network of devices might contain sufficient computational capacity to satisfy such needs, provided that capacity can be aggregated in an intelligent way. The first requirement to be addressed is thus *the impact of distribution*.

The question that needs to be answered is thus the following: *How is distributed processing achieved?* Whenever the sophistication of applications, particularly if dynamically deployed, reaches a point in which tasks, into which a computationally intensive job has been split, cannot be executed on a single node, they need to be dynamically offloaded in the network to others that could assist them with computation. The question is therefore how this can best be achieved.

Second Requirement: Impact of Network Conditions

Whenever distributed processing takes place, offload of information competes with the existing network congestion. Thus, whenever considerable amount of communication plays a role within the environment, it becomes crucial to decide to where to direct the offload and to analyse the impact of incorrect decisions. The second requirement to be addressed is thus *the impact of network conditions* on the distribution performance.

The question that needs to be answered is thus the following: *Towards which areas should a task offload be directed under heterogeneous network conditions?* In fact, whenever a task offload adds itself to existing heterogeneous network traffic levels, tasks cannot be randomly offloaded within the network but it becomes instead crucial to account for both task characterisation and real-time local network conditions in order to decide where the distribution should actually take place. Since the extreme dynamism of the environment from a communication perspective is difficult to emulate within simulators through models and representations of the radio physical and MAC layers, we perform an experimental evaluation in a WSN testbed. While traditional distributed systems are characterised by a stable network infrastructure where hosts are permanently connected to the network through high-bandwidth links, WSNs instead lack such reliable communication links and are more prone to congestion, collision, and interference. Therefore, the impact of network conditions within WSNs are likely to play a significant role.

1.3 Research Hypothesis

The scenarios presented in Section 1.1 illustrate a few examples of situations in which a set of computationally intensive applications must run together to allow systems to function effectively. To generalise this concept, let us define a number of intensive applications, namely Job_i with $i = \{1, \ldots, N\}$, required to run simultaneously in each scenario. Let us also assume that each Job_i has been split into M profiled tasks, namely $Task_{i,j}$ with $j = \{1, \ldots, M\}$ for the specific Job_i. We are now in a position to present the hypothesis behind our works as follows:

We posit that, by accounting for real-time local network conditions in distributing each $Task_{i,j}$, the overall Job_i latency decreases.

This implies that by distributing computationally intensive applications making informed decisions about the local level of communication within the environment, the *latency* in executing the whole Job_i decreases. We will test our research hypothesis along the latency dimension since, as described in Chapter 2, timeliness of information is vital for the scenarios we elected to use.

1.4 System Assumptions

In meeting our research hypothesis, the assumption that our system makes is as follows:

- *We assume the presence of an external task profiler able to characterise each $Task_{i,j}$.*
  Our system assumes the presence of an external profiler, either static or dynamic, that is able both to identify tasks $Task_{i,j}$ into which Job_i has been split and to characterise them both from a computational and a communication perspective. We believe that this assumption does not constraint the validity of the work, because it allows the system to choose and adopt the most suitable profiler to characterise tasks without affecting the distribution process.

1.5 Experimental Assumptions

In order to understand the statistical validity of the experimental results presented in this dissertation on the performance of deployed systems, it is crucial to highlight the experimental assumptions made during the testing as follows:

- *We assume constant radio connectivity over the timescale of each individual task offload.*
  We assume that system nodes are moving sufficiently slowly relative to one another so that radio connections are unlikely to break before offloaded tasks reach completion and consequent computed results are uploaded back. We believe this assumption is reasonable because the tasks involved in the scenarios described above are dynamic and latency matters.

- *We assume that nodes are homogeneous.*
  For the whole evaluation conducted on deployed testbeds, the nodes used in the system have identical computational capabilities, and thus our system does not make any assumption about the usage of devices required to be more computationally powerful than others. We believe this assumption is reasonable because it cannot always be assumed that environments are equipped with devices more computationally capable than others.

- *We assume that tasks have equal priority and are executed in parallel.*
  Every task into which a job is split has the same execution priority, thus we assume that task execution cannot be interrupted because of different task priorities. Moreover, because of concurrent execution, independent tasks can simultaneously and concurrently run on the same node. We believe this assumption is reasonable because task independence and parallel execution are common requirements of parallel computing.

- *Heterogeneous network contention levels are obtained through the introduction of additional interfering nodes.*

  In order to reproduce heterogeneous, unbalanced level of network communication, we equip the system with nodes generating additional network traffic perturbations. We test our approach along a range of settings and configurations and, for each result, we highlight the boundaries of the benefits brought by our approach. Interfering nodes emulate both general interference and that due to traffic aggregation at sink nodes. We believe this assumption is reasonable because, for the scenarios we investigate, heterogeneous network contention is envisioned to play a non-negligible role [Desch, 2001].

- *Sink nodes are disregarded in the experimental evaluation.*

  In the motivating scenario of this dissertation, sink nodes bridge the information flow between indoor and outdoor entities. They are therefore nodes affected by non-negligible network traffic. Whenever the result of a job computation is ready to be delivered to external authorities, information flows with routing protocols through sink nodes. Hence, an extra routing time would add itself to the actual job computation time if one were interested in measuring the global time spent from application definition to final result delivery. However, since our research hypothesis focuses on locally analysing the impact of network conditions whenever node collaborations become necessary to achieve job completion, the data we are interested in analysing is transparent to the presence of sinks (i.e. routing is a system invariant being required in both centralised and distributed scenarios). Thus, within the experimental evaluation we omit the routing time computation and we use, instead, the introduced interferer nodes to emulate the presence of sinks traversed by considerable traffic routes.

- *Jobs characterisations fall into specific categories.*

  Whenever task distribution becomes necessary to support a node in its need to achieve job completion, it is crucial to provide a characterisation for jobs to be distributed. The experiments performed and described in this dissertation do not aim to cover the whole range of possible situations. Instead, we analyse behaviours for a number of applications falling into specific sets of categories. In particular, initially we seek to distribute homogeneous jobs, then we heterogeneously profile them according to their different level of computational and communication load. Job profiling has been driven by adapting to our scenario task characterisations from the computational grids field [Foster et al., 2001, Raimondi et al., 2008]. We believe this assumption is reasonable because we provide experimentation for a wide range of tasks while, at the same time, characterising them according to models extracted from real-world data.

- *Errors in job profiling characterisations are disregarded.*

  In the experimental evaluation, we omit the introduction of errors in job profiling characterisations for the following set of reasons. Firstly, some of the heuristics are completely transparent to task profiling, thus a wrong characterisation would not impact on the decision making process.

Secondly, for those heuristics that, instead, account for profile characterisation, the obtained results do not outperform those obtained with the heuristic without profile characterisation. Therefore, we disregard the introduction of a theoretical error model within job profiling characterisations.

1.6 Contributions

Whenever computationally intensive applications cannot be executed on individual nodes, they need to be distributed and thus collaboratively executed. As mentioned above, the aim of this dissertation is thus to empirically investigate the impact of local network conditions on distribution within deployed systems. This is achieved by designing algorithms and mechanisms both to distribute jobs and to simultaneously cope with local network traffic in order to achieve performance improvements in terms of average job execution latency.

The research contributions tackled in the remainder of this dissertation can thus be organised according to the main experimental evaluation we have performed. We are now in a position to list the experimental research contributions as follows:

1.6.1 Distributed Wireless Ad-hoc Grid Paradigm

The first contribution of this dissertation is represented by the presentation of the *Distributed Wireless Ad-hoc Grid* (DWAG) paradigm [Rondini and Hailes, 2007a]. Firstly, we present the motivating scenario. Secondly, we detail the DWAG paradigm devised to allow computationally intensive applications to be collaboratively computed on resource-constrained devices. Thirdly, we list the requirements for applications to allow DWAG applicability. Finally, we describe DWAG main system flow, thus the sequence of actions undertaken through DWAG execution.

1.6.2 Network Conditions Impact on Distribution in Deployed Systems

The second contribution of this dissertation is represented by the empirical investigation of the effects of network traffic information on the performance of distributing homogeneous applications [Rondini and Hailes, 2007b, Rondini et al., 2008b]. Firstly, we propose two load sharing algorithms, namely the *Auction* and the *Lookup List* algorithms, to collaboratively distribute computationally intensive jobs. Secondly, we implement such algorithms on deployed WSNs testbeds composed of TMote Sky [TELOSB, 2010] devices. Thirdly, we empirically investigate the effects of local network conditions on the job execution performance. Hence we devise and integrate within the load sharing algorithms a *Bandwidth-Aware Task Scheduling* (BATS) offload mechanism that, dealing with both nodes computational capabilities and local network conditions, investigates the impacts of making informed offload decisions. Finally, we evaluate the effects of network conditions on the system performance at the presence of diversified traffic levels. The set of experiments has been specifically chosen to measure the impact of network conditions in settings where diversified level of network contention occurs but the offloaded tasks are homogeneous.

1.6.3 Impact of Heterogeneous Applications Distribution

The third contribution of this dissertation is represented by the empirical investigation of the effects of combining network contention information with task profiling characterisation while distributing and collaboratively executing heterogeneous applications kinds [Rondini and Hailes, 2010]. Thus, diverse sets of applications are locally distributed and collaboratively executed, with some of them requiring more computational capacity, while others more communication effort. Firstly, we define a set of application classes and we profile them according to their computational and communication requirements. Secondly, we empirically emulate heterogeneous job mixtures to be distributed. Thirdly, we evaluate the robustness of the BATS mechanism against a *Profile-Bandwidth-Aware Task Scheduling* (P-BATS) decision-making criteria combining task characterisation and network conditions. Fourthly, we devise a heuristic algorithm able to greedily compute an analytical lower-bound against which to compare the experimental data. Finally, we investigate the effects of network conditions, emphasising the consequent algorithmic performance degradation. The set of experiments has been specifically chosen to measure the impact of network conditions in settings where diversified level of network contention occurs but the offloaded tasks are heterogeneous.

1.6.4 Localisation Case Study

The fourth contribution of this dissertation is represented by the applicability of the devised DWAG paradigm with network control mechanism to a case study involving a localisation application [Rondini et al., 2008a]. Firstly, we describe the existing localisation algorithm proposed by Li et al. [Li and Kunz, 2009, Li, 2008, Li and Kunz, 2007a, Li and Kunz, 2007b], we study its limitations, and we propose adaptations to allow its applicability to WSNs testbeds. Secondly, we identify the set of tasks into which the localisation job can be split. Thirdly, we integrate and apply the load sharing algorithm with distribution mechanism to the localisation job. Since most of the work done so far in WSNs has almost exclusively been evaluated through simulations running on powerful machines, we analyse the real-world issues met by applying distributed algorithms on testbeds. Finally, we evaluate the system performance and the improvements achieved by accounting for network conditions during the decision-making phase.

1.7 Publications

The main contributions of this dissertation have been published in the papers listed below. All papers are co-authored by my supervisor, Professor Stephen Hailes, and several of them, by Dr. Li Li from Communications Research Centre, Canada, thanks to the intense research collaboration carried on while developing its collaborative localisation [Li and Kunz, 2009] approach, thus porting it from simulations within pragmatic deployments.

- E. Rondini, S. Hailes and L. Li. *Distributed Wireless Ad hoc Grids With Bandwidth Control For Collaborative Node Localisation*. In Proceedings of the 27^{th} annual IEEE/Boeing Military Communications Conference (MILCOM 2008), pages 1-8, San Diego, California, November 2008.

- E. Rondini, S. Hailes and L. Li. *Load Sharing and Bandwidth Control in Mobile P2P Wireless Sensor Networks*. In Proceedings of the 5^{th} IEEE International Workshop on Mobile Peer-to-Peer Computing (MP2P 2008) in conjunction with the 6^{th} IEEE International Conference on Pervasive Computing and Communication (PerCom 2008), pages 468-473, Hong Kong, China, March 2008.

- E. Rondini and S. Hailes. *Distributed Computation in Wireless Ad Hoc Grids with Bandwidth Control*. In Proceedings of the 5^{th} ACM Conference on Embedded Networked Sensor Systems (SenSys 2007), pages 437-438, Sydney, Australia, November 2007.

- E. Rondini and S. Hailes. *A Contiki-based Prototype for Creating Wireless Ad Hoc Grids*. Contiki Hands-On Workshop 2007, SICS, Kista, Stocholm, Sweden, March 2007.

1.8 Thesis Outline

The remainder of this dissertation is organised as follows:

Chapter 2 illustrates the common requirements of applications envisioned to run in the set of scenarios presented in Chapter 1. Our position to related work is then discussed, highlighting the most common assumptions in related work and enhancing the crucial importance of dealing with radio communications issues.

Chapter 3 presents the work motivating scenario together with the DWAG paradigm and illustrates the applicability requirements to allow jobs to be collaboratively computed. Our position to related work is discussed, highlighting the most common assumptions made.

Chapter 4 introduces the impact of network conditions on distribution, it describes the proposed load distributing algorithms, and it presents the bandwidth-aware BATS mechanism. An extensive experimental evaluation is carried on to test the impact of distributing homogeneous applications through our proposed approach on deployed testbeds. A critical evaluation of the achieved results concludes the chapter.

Chapter 5 discusses the impact network conditions in simultaneously distributing heterogeneous kinds of applications. It defines an additional P-BATS offloading mechanism accounting for a combination of task profile and local radio communication information to handle task distribution. Extensive experimental evaluation is performed on deployed testbeds with a variety of experimental settings and traffic configurations. A critical evaluation of the achieved results terminates the chapter.

Chapter 6 describes the localisation case study, presents the adaptations brought to allow both its distributed development on deployed WSNs testbeds and its DWAG applicability. Critical experimental evaluation concludes the chapter.

Chapter 7 summarises our work, points out its limitations, and identifies areas of future research.

Chapter 2

Requirements and Background

In Chapter 1, we provided a general overview of the research problem, its motivation, the assumptions and the contributions we plan to include in this dissertation. This chapter illustrates, instead, the requirements common to application scenarios presented in Section 1.1 and we provide an extensive critical analysis of the existing related work on such challenges.

The remainder of this chapter is thus organised as follows: (i) in Section 2.1, we analyse the main requirements of application scenarios listed in Section 1.1; and (ii) in Section 2.2, we detail the existing related work tackling such requirements.

2.1 Scenario Requirements and Background

In Section 1.1, we presented a set of possible application scenarios, such as emergency (Section 1.1), tracking and surveillance (Section 1.1), intelligent transportation systems (Section 1.1), taking advantage of Wireless Sensor Networks (WSNs) pervasive deployment and collaboration. We are now in a position to identify the set of requirements common to such scenarios, as follows:

- In real-world situations, it is frequently the case that multiple computationally intensive applications are required to run simultaneously. Although the current research trend is that of instrumenting the environment with small devices that are cheap enough to be pervasively deployed but sufficiently powerful to undertake basic computations, each resource-constrained device, alone, might not able to perform the totality of the required computationally intensive applications. In addition, although the computational capabilities of devices are growing quickly, their CPUs, memory size, as well as network bandwidth and latency, will continue to lag several orders of magnitude behind their fixed counterparts, thus it is unsafe to assume that each application will run entirely on a single device. Therefore, *distributed processing* becomes a system requirement within such scenarios. In Section 2.2.1, we describe related work concerning WSNs and the way in which distributed and collaborative processing has been tackled.

- Whenever distributed processing becomes a system requirement, devices are expected to synchronise and communicate with each other. Because a wired network is usually not available in pervasive deployments, communication between devices needs to be wireless. In wireless networks, each communication exchange modifies the actual level of traffic and leads to inevitable

heterogeneous environmental network conditions. Moreover, this additional network traffic adds itself to existing network communications. In fact, while traditional distributed systems (e.g. computational grids) are characterised by stable network infrastructures where hosts are permanently connected to the network through high-bandwidth wired links, WSNs lack instead of such reliable communication and are more prone to congestion, interference, message collisions and consequent retransmissions. Therefore, the analysis of *network conditions* becomes a system requirement within such scenarios. In Section 2.2.2, we describe the problems arising from radio communications in WSNs.

- The scenarios described in Section 1.1 are examples of real-world situations. However, although motivated by real-world scenarios, existing work in WSNs is often almost exclusively evaluated through simulations. Although simulators represent a straightforward choice for evaluations because of their extreme flexibility, modularity, scalability and adaptability, they make many simplifying assumptions, especially regarding wireless radio communication. In fact, the dynamism of the environment from a communication perspective cannot be fully matched through radio communication models and representations. For this reason, since simulator assumptions could compromise feasibility and actual performance of the devised algorithms, we decided to perform the entire evaluation of the current dissertation through experimentations on real-world deployments. Therefore, the comparison between *simulation vs. deployed system* experimentation becomes a requirement within such scenarios. In Section 2.2.3, we describe the most commonly used Operating Systems (OS) and platforms adopted to devise algorithms and infrastructures in WSNs.

We are now in a position to present the existing related work tackling the scenario requirements mentioned above.

2.2 Related Work

It is not the aim of this dissertation to provide a detailed literature review in the entire area of WSNs, as this would require the discussion of issues such as message routing and dissemination, security, quality of service and so on, aspects that have not been investigated within the current work. For a more exhaustive discussion on the state-of-the-art of WSNs, the interested reader may refer to [Karl and Willig, 2005, Stojmenovic, 2005], to name but a few examples.

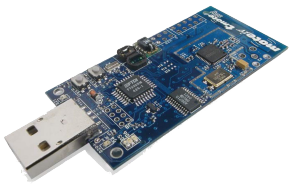
We are interested, instead, in discussing our position against the main scenario requirements listed in Section 2.1. In particular, we structure the discussion into three main parts, thus (i) distributed processing; (ii) network communication; and (iii) platforms in WSNs.

2.2.1 Distributed Processing in Wireless Sensor Networks

Recents advancements in MEMS have seen them gain wide popularity, especially because of the inexpensive production and pervasive deployment of miniaturised nodes with communication, computation, storage, and sensing capabilities.

As mentioned in Section 1.1, MICAz [MICAz, 2010], SunSPOT [SunSPOT, 2010], TMote Sky [TELOSB, 2010, Moteiv, 2006], Imote2 [Imote2, 2010] are just a few examples of the plethora

of embedded devices designed to perform one or more dedicated functions, often with real-time computing constraints. Despite their diverse form-factor and peculiar characteristics, the individual hardware components, used for processing and communication, fall into a small number of classes. Most platforms use a 16-bit Texas Instruments MSP430 processor or a 8/16-bit micro-controller of the Atmel ATMega family. Notable exceptions are the Imote2 and SunSPOT platforms, using the more powerful Intel PXA and ARM920T cores, respectively. The amount of volatile memory ranges from 2 KB to 512 KB, whereas external memory support varies from 128 KB to 4 MB. As for radio hardware, most platforms work in the 2.4GHz ISM band, and feature IEEE 802.15.4 [Baronti et al., 2007] compliant radio chips (e.g. the ChipCon CC2420 [Instruments, 2006]). Alternative solutions operate in the 433 or 868/916 MHz band (e.g. using the ChipCon CC1000 transceiver). Various types of sensors and actuators (e.g. temperature, sound, light, humidity, pressure) may be attached to the platforms by using expansion boards [EasySen, 2010] and thus the type of sensing/actuator device tends to be rather application-specific.



Tmote Sky

Figure 2.1: TMote Sky platform.

In the last few years TMote Sky devices, illustrated in Figure 2.1, have increased considerably their popularity and market share in both academia and industry [Basaran et al., 2006], thanks to their inexpensive and easy deployable general-purpose platform. For this reason and because of the opportunity to perform experimentation on a testbed, the Heterogeneous Experimental Network (HEN) [HEN, 2010] TMote Sky sensor testbed deployed at the Department of Computer Science at University College London (UCL-CS), we adopted TMote Sky devices for the experimental evaluation of the current dissertation. Thanks to the ultra-low-power micro-controller and the electrically buffered external components that can be switched-off when not in use, the platform achieves very low quiescent currents in the sleep mode, and very fast wake-up times. Moreover, TMote Sky devices are equipped with a MSP430F1611 micro-controller from the Texas Instruments MSP430 family of ultra-low-power 8MHz 16-bit micro-controllers. The on-board micro-controller offers 10KB RAM and 48KB flash programmable ROM. The board provides 1MB of external flash memory connected to the SPI bus. TMote Sky devices are equipped with ChipCon CC2420, an IEEE 802.15.4 compliant radio transceiver featuring data rates of up to 250kbps. The transceiver is connected to an onboard antenna providing about 50m range indoors and 125m outdoors. The TMote Sky board has predefined positions for mounting humidity and temperature sensors from Sensirion AG (i.e. SHT11 and SHT15 are the supported models), as well as for light sensors like the Hamamatsu Corporation S1087 for sensing photosensitive solar radiation or the S1087-01 for total spectrum measurements. TMote Sky are approximately 70mm long, 30mm wide and

20mm tall, and they weigh 23g excluding batteries.

To date, TMote Sky devices have been adopted within a wide spectrum of WSNs applications [Arampatzis et al., 2005, I. F. Akyildiz et al., 2002, Intanagowiwat et al., 2000, Hill et al., 2000, Kulik et al., 1999] ranging from traffic and wildlife habitat monitoring, battlefield surveillance, target detection and tracking, home automation, flood and fire detection, security and surveillance, emergency scenarios and military applications. Moreover, they have been mainly adopted with the idea of instrumenting the physical environment with resource-constrained, low cost devices able to monitor environmental phenomena by progressively gathering data readings enabling possible system actuators [Costa et al., 2007].

However, in situations like those presented in Section 1.1 in which several computationally intensive applications are required to be simultaneously in place, the amount of computation to be performed can necessitate greater computational capacity than is available on a single low-power processor. This has therefore motivated the implementation of distributed processing approaches in WSNs.

A comprehensive review of the existing literature on techniques and protocols for in-network aggregation and processing in WSNs has been presented [Fasolo et al., 2007]. In-network aggregation is the global process of gathering and routing information through a multi-hop network, processing data at intermediate nodes with the objective of reducing resource consumption (i.e. mainly with respect to energy), thereby increasing network lifetime. Classic routing strategies [Akkaya and Younis, 2005] are usually based on a hierarchical organisation of the nodes in the network. In fact, the simplest way to aggregate data flowing from the sources to the sink is to elect some special nodes which work as aggregation points and define a preferred direction to be followed when forwarding data. Regarding routing, some protocols are based on a tree structure (e.g. TAG [Madden et al., 2002], Directed Diffusion [Intanagowiwat et al., 2000], PEGASIS [Lindsey et al., 2002], DB-MAC [Bacco et al., 2004], TinyDB [Madden et al., 2005], etc.), while others on a cluster head (e.g. LEACH [Heinzelman et al., 2002], Cougar [Yao and Gehrke, 2002], etc.) one. The main advantage of a clustered structure is that it directly allows aggregation of data at the cluster head. Such algorithms work well in relatively static networks where the cluster structure remains unchanged for a sufficiently long time, but they may be fragile when used in more dynamic environments. Moreover, the cost required to maintain the hierarchical structure is substantial. Similar considerations apply to tree-based schemes.

In their work [Fasolo et al., 2007], the authors critically analyse several aspects of the existing approaches. Firstly, although in-network aggregation touches several layers of the protocol stack, and any efficient solution is likely to require a cross-layer design, most of the existing research focuses on issues such as routing, often considering only very simple approaches to aggregate data. Moreover, very few cross-layer solutions, accounting for application, data representation, routing and MAC aspects have been presented. Such schemes as are proposed mainly focus on a subset of such aspects, typically trying to merge routing and data aggregation techniques, but ignoring MAC, memory and computational issues regarding data aggregation processing. In fact, the authors stress the aspect that without a joint

design, the performance gained at the routing layer may be partially lost due to MAC radio inefficiencies. Hybrid algorithms [Manjhi et al., 2005], which combine the properties of different approaches, tune the degree of aggregation and may therefore facilitate the adaptation of the aggregation scheme. Thus, they are more suitable for the design of schemes that are able to adapt to application needs. However, while many theoretical approaches have been devised, very few of them have been turned into practice and thus applied to deployed WSNs.

Additional cluster-based techniques have also been developed to allow distributed processing [Shah et al., 2009, Abbasi and Younis, 2007, Singh and Prasanna, 2003, Xu and Qi, 2004, Qi et al., 2003, Fuggetta et al., 1998]. However, they all share the view that there is a hierarchical network architecture comprising of a high number of low-cost, less powerful sensors, and a small number of higher-cost, more powerful cluster heads. Such algorithms are applicable if the structure of the deployed network is indeed hierarchical, and if the geographic distribution of cluster heads relative to sensor nodes is appropriate. However, cluster-based approaches are inappropriate otherwise, and the existing approaches also fail to take into account issues that relate to bandwidth utilisation, message collision and realistic radio modelling: communication is mainly assumed to be collision-free. Furthermore, some of them [Xu and Qi, 2004, Qi et al., 2003] make two strong assumptions: firstly, when a certain event occurs, all sensor nodes can detect it and collect raw data, so that there is always the same number of sensor nodes participating in data fusion; secondly, there are no events simultaneously occurring in the field. Such assumptions are limiting to the generality of the approach as is, more generally, the lack of testing outside the simulation domain.

Moreover, further surveys [Kuorilehto et al., 2005, Abbasi and Younis, 2007] on application distribution in WSNs analyse the implemented distribution aspects in middleware and stand-alone protocol proposals. The result is that most of the devised approaches neither support possible task migration or allocation, nor remote task communication (e.g. MiLAN [Heinzelman et al., 2004], TinyDB, Cougar). In [Ruiz-Ibarra and Villasenor-Gonzalez, 2008], a framework for the classification taxonomy of coordination mechanisms designed for Wireless Sensor and Actor Networks (WSANs) environments is provided. Based on this taxonomy, a comparative analysis is presented to study some of the most representative coordination mechanisms [Shah et al., 2006, Yuan et al., 2006, Ngai et al., 2006, Melodia et al., 2006, Melodia et al., 2007] proposed for WSANs. In general, the proposed mechanisms proceed to split the event area and perform a hierarchical coordination by employing localisation information; this is done with the objective of selecting the proper nodes (i.e. sensors and actors) that will react in response to a specific event with the smallest possible response time. The proposed applications for each of the coordination mechanisms differ with respect to the frequency of the events. In general, the proposed coordination mechanisms try to comply with the support of real-time response requirements along with an efficient use of energy in the WSAN. Moreover, none of the coordination mechanisms implement a mechanism to guarantee data security and system robustness, task prioritisation, allocation and actual migration, and cross-layer designs to reduce the overhead in the network in energy consumption and latency. Finally, there is a generalised lack of system evaluation within deployed systems.

An approach designed to achieve an energy-efficient collaborative signal processing system has been presented in [Chiasserini and Rao, 2002]. This was reliant on the divide-and-conquer paradigm through which a problem is divided into multiple sub-problems of smaller dimensions. Each sensor, implementing the same function, exploits its processing capability to solve a sub-problem so that the final solution to the original problem is obtained by combining the partial results computed by every single node. The authors apply the idea of collaborative signal processing to implement a Distributed Digital Signal Processing (DDSP) approach to the Fast Fourier Transform (FFT) algorithm, giving a reduction in energy consumption and latency. The approach represents an example of computational distribution in a sensor network. However, it assumes the replication of the same algorithm on every single node, allowing the simple splitting of the data samples vector across different sensors. Moreover, the work lacks infrastructure and protocols dealing with actual resource allocation and task migration. Again, this is limiting as radio communication and node synchronisation are not taken into account.

The problem of task assignment and migration has been recently tackled within multi-agent systems [Vinyals et al., 2008]. Agilla [Fok et al., 2005] is one of the precursors in the use of mobile agents in middleware for WSNs. Its approach is to use software agents that can move from one node to another in the network. It also allows multiple agents to run on the same node. These characteristics provide the desired features of energy saving, as agents can run near the data avoiding unnecessary communication. In [Kho et al., 2007], a proposal to use a distributed mechanism to control adaptive sampling to support energy-constrained network operations is presented. In the proposal, each sensor is considered an autonomous agent, enabling decentralised control of the sampling rate of sensor nodes in the application domain of flood monitoring. Besides the contribution in the increase of the efficiency of the energy consumption, the goal is also to maximise the information value of the data collected to the base station. The network is homogeneous, the approach is domain-specific and all agents have an uniform role. AWARE [Erman et al., 2008] proposes a middleware whose goal is to provide integration of the information gathered by different types of sensors, including WSNs and mobile robots. Leung et al. [Leung et al., 2008] present an information-processing paradigm for intelligent sensor networks. Nodes in sensor networks have different levels of autonomy in terms of signal processing, information fusion and situation assessment in order to contribute to the overall system decision making. Their approach is based on the use of a genetic algorithm to provide learning features to the sensor nodes and fuzzy cognitive maps to perform situation assessment. The sensor networks aimed at by this work are those composed only of rich nodes, as the architecture and the techniques used are not lightweight enough to fit in low-end nodes.

Another approach involving task assignment in a network of resource-constrained computing platforms (microservers), was proposed by [Abrams et al., 2006]. An optimisation algorithm for the assignment of tasks (dedicated to tracking physical objects as they move) to microservers is studied. The tasks migrate within the network as the physical object moves, and the algorithm seeks to minimise the number of task migrations, whilst guaranteeing that the highest number of physical objects is monitored at all times. Two heuristics that account for the knowledge of future trajectories are devised. The proposed

approach provides an interesting example of forced task migration, even if energy and communication consumption requirements are not explicitly considered in this model.

Another approach, aiming to maximise the network lifetime while minimising job execution time, was presented by [Park and Srivastava, 2003]. The authors devise a three phase framework. In the first phase, a task graph is parsed, every task is decomposed, transformed and assigned to the sensor nodes in order to optimise a cost function (mapping of the total energy consumption of the network) using the simulated annealing technique. In the second phase, the task is scheduled in order to minimise collisions while communicating. After these two phases, the sensor network can perform its task (in the third phase): if a node reaches a low energy state, the tasks running on it need to migrate towards healthier neighbouring nodes. However, the approach makes several assumptions: (i) task decomposition, transformation and assignment are centralised; (ii) radio communication among sensors is established through minimal energy routes and radio channels are symmetric, a particularly dubious assumption in the real world.

In [de Freitas et al., 2009], a middleware is presented that aims at addressing mission-driven heterogeneous sensor networks deployed in highly dynamic scenarios. These scenarios require middleware reflection to support nodes autonomous behaviours, by using an agent-oriented approach, and to address changes in the environment and in the network. Multi-agent reasoning is used in order to set up, configure and reconfigure the network. Formal definition of the mission statements and conditions (described in MDL) are presented, as well as their mapping to the elements of a Belief-Desire-Intention (BDI) approach that supports the proposed network-wide reasoning. A supporting agent framework is presented, with the description of the mechanisms used for agent reasoning. However, the complexity of the approach makes it greatly unsuitable to run on resource-constrained devices.

In [Sensoy et al., 2009], a way is proposed to describe tasks semantically with their requirements and constraints so that software agents can reason about those tasks and determine what type of sensor resources they may need. Based on the semantic description and reasoning mechanisms, they propose a distributed agent-based approach to efficiently allocate sensor resources to tasks. Thus, Semantic Web technologies and multi-agent systems are integrated in a way to determine and allocate resources to tasks in WSNs. First, the approach proposes the adoption of an OWL-S process ontology with domain ontologies used to describe tasks semantically. Second, it combines ontological reasoning with logic programming to enable software agents to reason about the required sensor types for the tasks. Third, it present a decentralised agent-based approach to solve the sensor-task assignment problem. Task agents interact with sensor agents to allocate the most useful sensors for the tasks. For this purpose, task agents convert the allocation problem into a knapsack problem utilising the multi-round knapsack algorithm to provide a concrete solution.

For all the approaches mentioned above, not only is the sophistication of the proposed algorithms non-negligible, but also the devised techniques have been not effectively tried on resource-constrained devices. Evaluation has been performed through simulations, which are thus unable to provide a faithful snapshot of the complex reality of radio wireless communications. Moreover, within multi-agents

systems, the distribution is implemented through the movements of software agents simulated with high-level programming languages not practically suitable or applicable within resource-constrained devices.

Finally, an interesting contribution aiming at the resolution of the resource allocation problem by distributing tasks at run-time using a real-time approach is the one presented in [Giannecchini et al., 2004]. The authors consider systems, such as WSNs, in which total CPU and bandwidth cannot be considered constant all the time. To achieve this goal, the authors propose a fast online resource allocation algorithm (CoRAl) to reconfigure a sensor network dynamically, whenever a new hot spot occurs or node activity changes, while taking into account the timing constraints of end-to-end tasks. An end-to-end task is a chain of ordered subtasks, each of which is executed on different nodes in a pipeline-fashion (the latter task cannot start the computation until the former has finished). The scheduling of task activity is treated as an optimisation problem assuming the usage of Earliest Deadline First (EDF) for the nodes' scheduling algorithms and of implicit EDF for the wireless network MAC protocol. However, the authors assume the use of FDMA-TDMA, in which all network nodes are synchronised on a frame basis and a single data packet of constant size is sent during each frame. They solve the hidden terminal problem by assuming a network of fully connected nodes. The model of the wireless channel is simulated through a dummy node capable of sending messages periodically. Starting from the above approach, Liu et al. in [Liu et al., 2006] solve some of the previous drawbacks (e.g. replacing FDMA-TDMA with RICH schema), with the aim of studying centralised and distributed solutions to the optimal sampling rate assignment problem in Real-Time Wireless Sensor Networks (RTWSN), by formalising them as non-linear optimisation problems.

Consequently, analysing the existing related work in the field of distributed and collaborative processing in WSNs, we notice that: (i) most of the approaches rely upon hierarchical, predefined physical structures of nodes; (ii) they often model task distribution as optimisation problems, but they do not practically test them on deployed systems; (iii) they disregard the actual impact of radio communication layer, used by nodes to synchronise with each other; (iv) they devise approaches with the idea of minimising node communications while, at the same time, not practically accounting for possible existing sources of interference; (v) they are often too sophisticated to run on resource-constrained devices.

What is missing: A simple distributed solution allowing nodes to collaborate in an ad-hoc manner with each other to facilitate the execution of computationally demanding applications.

2.2.2 Network Communication Issues in WSNs

Whenever distributed processing becomes a system requirement, devices are expected to synchronise and communicate with each other by wirelessly exchanging messages. Each communication exchange modifies the actual level of traffic and leads to inevitable heterogeneous environmental network conditions. Moreover, this additional network traffic adds itself to existing network communications. In fact, while traditional distributed systems (e.g. computational grids) are characterised by stable network infrastructures where hosts are permanently connected to the network through high-bandwidth wired links, WSNs lack instead of such reliable communication and are more prone to congestion, interference, message collisions and consequent retransmissions.

Two fundamental problems that arise from wireless radio communications are explained as follows:

- *Hidden terminal problem:* Nodes in a wireless network that are out of range of a collection of other nodes are called hidden nodes. As illustrated in Figure 2.2, assume that we have a physical star topology with a wireless node A surrounded by a set of additional nodes B and C. Surrounding nodes B and C fall into A's communication range, but they cannot hear and communicate with each other, because of the lack of a physical connection among them. Thus, B is likely to hear A but unlikely to hear C (i.e. B is a hidden node with respect to C). Similarly, C is likely to hear A but unlikely to hear B (i.e. C is a hidden node with respect to B). In such situations, a problem arises when nodes B and C simultaneously send message packets to A. Since node B and C cannot sense the carrier, the CSMA/CA mechanism does not push devices to back off and thus message collisions occur as an effect of the hidden terminal problem [Kleinrock and Tobagi, 1975, Moh et al., 1998].

  To overcome this problem and thus to reduce frame collisions introduced by the hidden terminal problem, handshaking mechanisms such as Request to Send/Clear to Send (RTS/CTS) have been implemented in conjunction with the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) scheme [Colvin, 1983]. Thus, a node wishing to send data initiates the process by sending a Request to Send frame (RTS). The destination node replies with a Clear to Send frame (CTS). Any other node receiving the RTS or CTS frame should refrain from sending data for a given time, thus tempting to solve the hidden node problem. The amount of time the node should wait before trying to get access to the medium is included in both the RTS and the CTS frame. This protocol was designed under the assumption that all nodes have the same transmission range. RTS/CTS is not a complete solution to the hidden terminal problem, therefore additional adaptive acknowledgment (ACKs) mechanisms have been studied and combined to contain its effects.

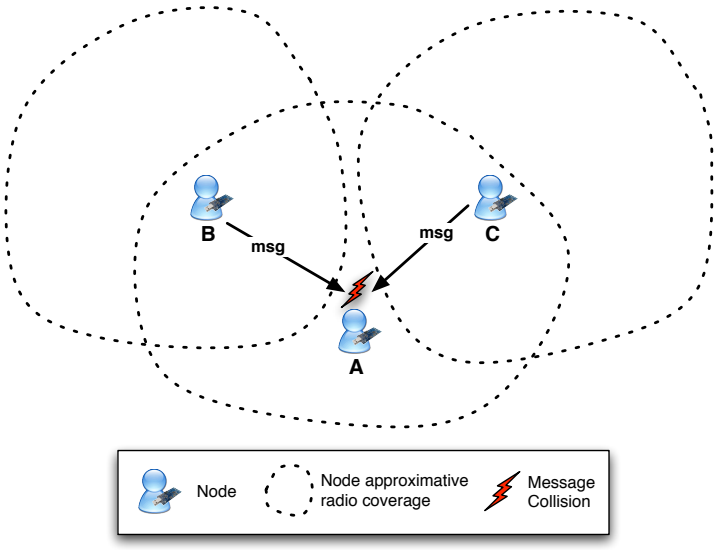


Figure 2.2: Hidden terminal problem in wireless networks.

- *Exposed terminal problem:* In wireless networks, the exposed terminal problem occurs when a node is prevented from sending packets to other nodes due to a neighbouring transmitter. As illustrated in Figure 2.3, assume we have four nodes, labeled S_1, R_1, S_2 and R_2, where the two receiver nodes R_1 and R_2 are not in range of each other, while the two sender nodes are in range of each other. Thus, if a message exchange is performed between S_1 and R_1, node S_2 is prevented from transmitting to R_2 since it concludes, after applying CSMA/CA, that it will interfere with the transmission initiated by its neighbour S_1 (although R_2 could still receive the transmission of S_2 without interference). This thus generates the exposed terminal problem [Jayasuriya et al., 2004].

  Once again, IEEE 802.11 RTS/CTS mechanism helps solving the problem, but only if nodes are synchronised and packet sizes and data rates are the same for both transmitting nodes. Thus, when a node hears an RTS message from a neighbouring node, but not the corresponding CTS, that node assumes the presence of an exposed terminal problem and, therefore, it transmits to its neighbouring nodes.

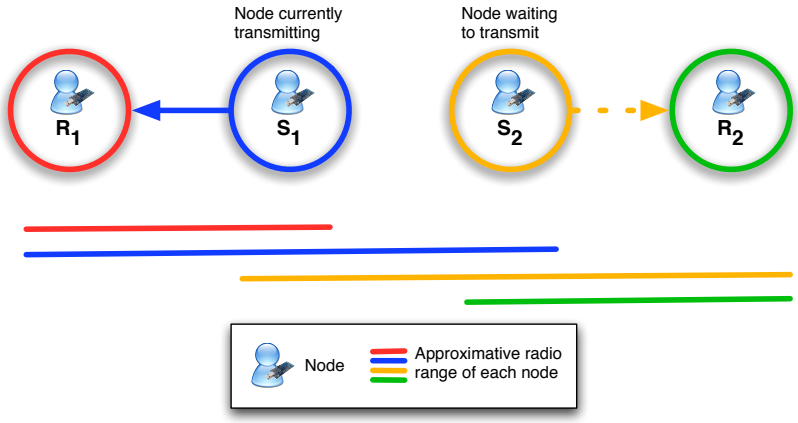


Figure 2.3: Exposed terminal problem in wireless networks.

Whenever devices are deployed within testbeds and are required to collaborate with each other, such interactions are affected by two sources of network traffic:

- *Interference at MAC layer:* Whenever multiple nodes simultaneously and *actively* communicate with each other, message collisions play a role. However, if the communication is performed within the same MAC layer (e.g. IEEE 802.15.4/ZigBee), the protocol automatically deals with possible message collisions by forcing devices to back off [Bertocco et al., 2008]. Thus, whenever multiple devices aim at simultaneously occupying the shared radio medium, IEEE 802.15.4 deploys the CSMA/CA mechanism, through which a station wishing to transmit first determines the status of the channel. In the case of channel status idle then the station is allowed to transmit, otherwise, in the case of channel status busy, it waits until the channel is free again.

- *Interference at physical layer:* Whenever nodes belong to a deployed system, they are *passively* affected by the existing network contention generated by additional sources of communication adopting different MAC (e.g. IEEE 802.11b/WiFi, IEEE 802.15.1/Bluetooth,

etc.) [Bertocco et al., 2008, Shina et al., 2007]. IEEE 802.15.4 systems with a CSMA/CA mechanism can thus be affected by radio interference at physical layer when a WSN terminal and a nearby in-channel interfering source transmit simultaneously. In such situation, devices cannot themselves minimise message collisions through the MAC layer. Instead, existing network traffic passively affects message exchanges from devices thus increasing the amount of congestion, interference, message collisions and consequent retransmissions. Moreover, the simultaneous transmission of multiple signals, and the subsequent superposition, provokes data packet collisions at the receiver node, and, consequently, losses in terms of quality parameters like error vector magnitude (EVM) and packet error ratio (PER) [Proakis, 2000].

In order to cope with the issues mentioned above together with maximisation of the network lifetime, scalability, collision avoidance and tolerance to network changes, a plethora of MAC layer protocols have been hitherto proposed [Naik and Sivalingam, 2004, Demirkol et al., 2006] (e.g. S-MAC/T-MAC/DSMAC, WiseMAC, TRAMA, Sift, DMAC, etc.). They essentially fall into two categories. Randomised protocols such as S-MAC [Ye et al., 2002, Ye and Heidemann, 2003, Ye et al., 2004], B-MAC [Polastre et al., 2004] and T-MAC [van Dam and Langendoen, 2003] regulate the access to the physical layer opportunistically, based on the current transmission requests. Conversely, time-slotted protocols such as TRAMA [Rajendran et al., 2003, Rajendran et al., 2006], FDMA, TDMA, CDMA and others assign the nodes with predefined time-slots to schedule their transmissions over time. While the former class of protocols is easier to implement and better tolerates nodes joining or leaving, the latter class enables better reliability and greater energy savings. The radio can also be switched to a low-power mode based on the current transmission schedule. Nonetheless, the latter protocols generally require tight time synchronisation among the nodes in some k-hop neighbourhood. However, despite the plethora of devised protocols, there is a generalised lack of a uniquely accepted standard.

Future directions have been discussed in [Ali et al., 2006]. In particular, the critical analysis of the proposed techniques has lead to state that the vast majority of related work in the MAC protocols area has, to date, almost exclusively focused on the issue of energy optimisation, while instead it underestimates traditional goals such as fairness, delay and real bandwidth utilisation. In particular, the authors [Ali et al., 2006] propose a set of general future directions in the MAC area listed as follows: (i) energy efficiency criteria should not be the only evaluating metric for a MAC protocol, instead optimisation criteria for latency, reliable data delivery and compliance with real-time constraints should be met; (ii) sensor networks applications exhibit specific traffic patterns that should be carefully designed in MAC protocols; (iii) security issues (e.g. against eavesdropping and malicious behaviours) should be taken into account; (iv) different MAC protocols, operating at the same frequency band should be able to coexist in the same physical environment; (v) MAC protocols, designed to be applied to static scenarios, fail to provide acceptable performances when applied in mobile environments, thus MAC protocols should address node mobility requirements; (vi) to obtain more realistic insight into MAC layer performance, researchers should move from simulation environments to prototypes (software defined radios) and real-world experiments.

What is missing: A cross-layer solution pragmatically accounting for local network conditions in performing distributed processing.

2.2.3 Operating Systems for WSNs

The design of Operating Systems (OS) for WSNs deviates from traditional OS design due to the peculiar characteristics of WSN devices like constrained resources, high dynamics and inaccessible deployment. Therefore, devices must be programmed though specific lightweight OS. Complete surveys [Reddy et al., 2009, Basaran et al., 2006], detailing the main characteristics of the existing OS for WSNs, have recently been published. Architecture, execution model, reprogramming, scheduling and power management are the main OS features chosen in [Reddy et al., 2009] to classify the existing WSNs OS.

The most widespread OS solution for WSNs is TinyOS [Hill et al., 2000, System, 2010b]. Alternatives are the Contiki OS [Dunkels et al., 2004, System, 2010a], SOS [Han et al., 2005], Mantis [Abrach et al., 2003], RETOS [Cha et al., 2007] and NANO-rk [Eswaran et al., 2005]. TinyOS has a monolithic architecture since it uses a component model at compile-time and single static image at run-time. Mate' [Levis and Culler, 2002], built on top of TinyOS, and CVM (Contiki VM), built on top of Contiki, provide virtual machine architectures. Mantis, Contiki OS and SOS use modular approach. Among them, SOS, Mantis, and Contiki feature dynamic linking functionality. The concurrency model employed varies from event-driven solutions [Hill et al., 2000] to preemptive, time-sliced multi-threading [Abrach et al., 2003, Cha et al., 2007] and asynchronous message passing [Han et al., 2005]. These approaches have also been further augmented in several works. Examples are Fibers [Welsh and Mainland, 2004], TinyThreads [McCartney and Sridhar, 2006], and Y-Threads [Nitta et al., 2006]. These approaches add a threading model to TinyOS by enabling different forms of blocking calls. Fiber allows a single blocking context, whereas TinyThreads enables cooperative multi-threading by giving programmers a way to yield explicitly the current execution context. Y-Threads, instead, are similar to Fiber, but they provide preemptive multi-threading. Protothreads [Dunkels et al., 2006] enable a form of cooperative multi-threading in Contiki. However, they do not store the execution contexts, thus requiring the programmer to save and restore the relevant state by hand. The Object State Model [Kasten and Römer, 2005] extends the event model with state machines. It also provides the ability to compose different state machines to build hierarchies.

Let us now focus on the two main OS running on TMote Sky devices, namely TinyOS [Hill et al., 2000, System, 2010b] and Contiki OS [Dunkels et al., 2004, System, 2010a].

TinyOS is one of the first environments designed to meet the requirements of resource-constrained, event-driven and networked embedded systems. Originally developed by the University of California, Berkeley and Intel at the beginning of the last decade, it has since then become the most popular OS for WSNs. Many of the features of TinyOS stem from the design goals of its implementation language called NesC, an extension of C adapted to cope with the special needs of networks of embedded devices. One of the main characteristics of its programming model is the use of component-based modularisation. In this model, the functionality of the traditional monolithic abstraction layers is broken-up into smaller,

self-contained building blocks that interact with each other via clearly defined interfaces. This hiding of the implementation behind these interfaces preserves the modularity of the solution and promotes reuse. At the same time, the component model supports interactions between the building blocks. The TinyOS concurrency model is based on commands, asynchronous events, deferred computation called tasks and split phase interfaces. The function invocation (i.e. command) and its completion (i.e. event) are separated into two phases in interfaces provided by TinyOS. An application user has to write the handler which should be invoked on the triggering of an event. Commands and event handlers may post a task, which is executed by the TinyOS FIFO scheduler. Tasks are non-preemptive and run to completion. However, tasks can be preempted by events but not by other tasks. Data race conflicts that arise due to preemption can be solved using atomic sections. The communication architecture of TinyOS uses the concept of Active Messages (AM). AMs are small packets of size 36 bytes and a one byte handler ID. A node, after receiving an AM, dispatches it to corresponding registered handlers. TinyOS's purely event-driven model has obvious disadvantages like low programming flexibility and non-preemption.

In the last couple of years, TinyOS has mainly competed against Contiki OS for TMote Sky programmability. Contiki OS is an open-source, highly portable, networked, multi-tasking OS for memory-constrained networked embedded systems. A typical Contiki configuration requires 2KB of RAM and 40KB of ROM. Contiki OS has been developed at Swedish Institute of Computer Science (SICS) in Sweden and it has recently been adopted within a number of projects (e.g. RUNES [Costa et al., 2007], [Pasztor et al., 2010]). Contiki OS combines the advantages of both event and thread models. In fact, it is primarily an event-driven model, but also supports multi-threading as an optional application level library. In fact, the event driven kernel does not provide multi-threading by itself. Instead, preemptive multi-threading is implemented as a library that can optionally be linked with applications. This library consists of two parts: a platform independent part, which is the same for all platforms on which Contiki OS runs, and a platform specific part, which must be implemented specifically for the platform that the multi-threading library should run. Although an external library (i.e. $mt.c$) has been implemented to allow multi-threading at the application level, its usage leads to non-negligible memory overhead when loaded on resource-constrained devices. Thus, its adoption is usually replaced by the more lightweight protothreads. Contiki OS includes also advanced reprogramming support in the form of loadable modules. This feature is useful for both program development, since it shortens the development cycle, and system deployment. In fact, replacing a module is more energy-efficient than replacing the whole image.

Contiki OS supports, in fact, three concurrency models: events, threads and protothreads. The Contiki OS kernel is event-based, but application programs can use any of the three concurrency models, or a combination of them. The kernel consists of a lightweight event scheduler that dispatches events to running processes and, periodically, calls processes polling handlers. Program execution is triggered either by events dispatched by the kernel or through the polling mechanism. The kernel does not preempt an event handler once it has been scheduled. Therefore, event handlers must run to completion. In addition to the events, the kernel provides a polling mechanism. Polling is seen as a high priority event sched-

uled in-between each asynchronous event. Polling is used by processes that operate at hardware level to check for status updates of hardware devices. When a poll is scheduled, processes that implement a poll handler are invoked, in order of their priority. The Contiki OS kernel uses a single shared stack for the entire process execution. The use of asynchronous events reduce stack space requirements as the stack is rewound between each event handler's invocation. Thus, events are classified as asynchronous and synchronous. Synchronous events are scheduled immediately while asynchronous events are scheduled later. Polling mechanism is used to avoid race conditions. In Contiki OS everything (i.e. communication, device drivers, sensors data handling) is implemented as service. Each of the service has interface and implementation. Application is aware of only interfaces.

Protothreads [Dunkels et al., 2006, Dunkels and Schmidt, 2005, Dunkels et al., 2005] are a novel programming abstraction that provides a conditional blocking wait statement, PT_WAIT_UNTIL(), that is intended to simplify event-driven programming for memory-constrained embedded systems. The operation takes a conditional statement and blocks the protothread until the statement evaluates to true. If the conditional statement is true the first time the protothread reaches the PT_WAIT_UNTIL(), the protothread is not blocked but continues to execute without interruption. A protothread is stackless, meaning that it does not have a history of function invocations. Instead, all protothreads in a system run on the same stack, which is rewound every time a protothread blocks. This mechanism is extremely advantageous in memory constrained systems, where a thread stack might use a large part of the available memory. A protothread runs inside a single function and it is therefore activated each time the function is invoked. Since they are stackless, protothreads can only block at the top level of the function. Nevertheless, by using hierarchical protothreads, it is possible to perform nested blocking. Protothreads can thus be seen as a combination of events and threads. From threads, protothreads have inherited the blocking wait semantics. From events, protothreads have inherited the stacklessness. Finally, although Contiki does not provide explicit power management abstractions, it allows the application programmers to implement such mechanisms. One such provision is exposing its internal queue state to the applications. Thus, applications can decide to power down the system when there are no events to be scheduled. Consequently, the processor wakes up in response to an external event which is handled by an external poll handler.

Contiki contains two radio communication stacks: uIP TCP/IP and Rime. The uIP TCP/IP [Dunkels, 2003] stack allows Contiki OS to communicate over the Internet thus favouring interoperability with existing IP network infrastructures and systems. The uIP implementation is designed to have only the absolute minimal set of features needed for a full TCP/IP stack. Web server, ftp server as well as a number of other applications have been developed to test such interoperability. Although the uIP TCP/IP stack may not be as energy-efficient as custom protocols tailored for WSNs, its early adoption was motivated by the need to promote systems integration and communication. Only recently a novel communication stack, namely Rime [Dunkels et al., 2007], has been devised to cope with the increasing (i) network heterogeneity, (ii) number of link layers, (iii) MAC protocols, and (iv) underlying transportation mechanisms. Rime [Dunkels et al., 2007] is a lightweight communication

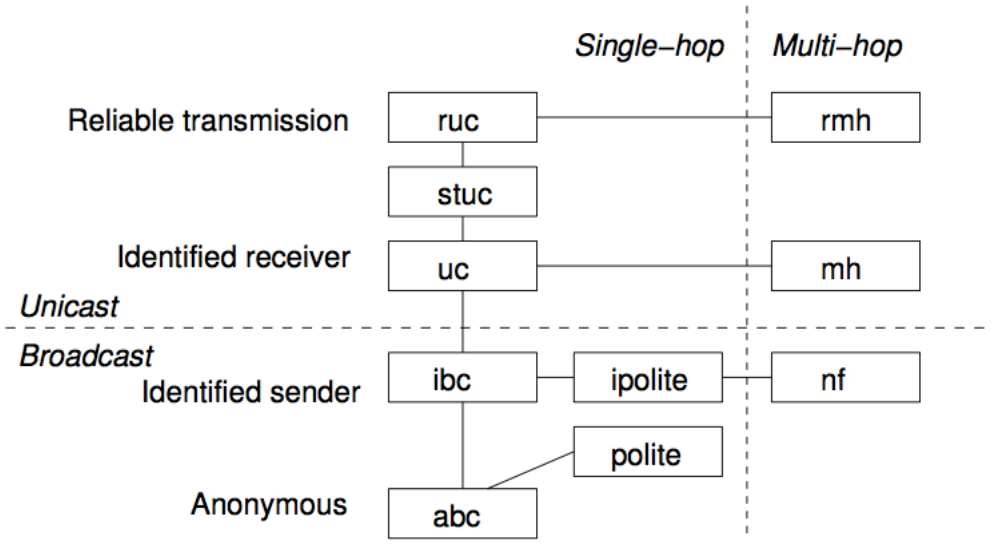


Figure 2.4: The communication primitives in the Rime stack and how they are layered.

stack designed for low-power radios. As illustrated in Figure 2.4, it provides a wide range of communication primitives and protocols, such as multi-hop data collection, multi-hop unicast mesh routing, and reliable multi-hop network flooding. As shown in Figure 2.5, applications and protocols transfer data from application layers down to the Rime stack. The Rime stack adds packet attributes to the application data before passing it to the underlying Chameleon header transformation module. This latter module builds packet headers from the packet attributes and sends the final packets to the link-level device driver or the MAC layer. The MAC layer can inspect the packet attributes to decide how the packet should be transmitted. For example, broadcast packets may be sent differently from unicast packets, and packets that need single-hop reliability can be sent with reliable link-layer acknowledgements turned on. The Rime stack is built around a lightweight layering principle: the communication primitives are designed in a layered hierarchical fashion, where more complex communication primitives build on the top of simpler ones. For WSNs, this lightweight layering principle has several benefits. First, as the communication primitives are simple, they are easy to implement and test. Second, the memory footprint of the implementations of the primitives is small, which is important for memory-constrained sensor nodes. Third, as applications may attach to any stack layer, they can precisely express the amount of communication features they need. In more heavyweight-layered stacks, such as the TCP/IP protocol stack, it is generally not possible to express such fine-grained feature requirements.

The inherent benefits brought by the Contiki OS (i.e. a concurrent and modular event-driven model supporting cooperative multi-threading and library dynamic linking) have motivated our choice of developing the algorithms presented in the current dissertation by adopting the Contiki OS over TinyOS (i.e. a monolithic purely event-driven architecture with low programming flexibility and non-preemption). Moreover, recall that tests will be performed adopting, and thus testing, both uIP TCP/IP and Rime radio communication stacks.

Finally, let us now drive a brief discussion about simulation environments. Early performance stud-

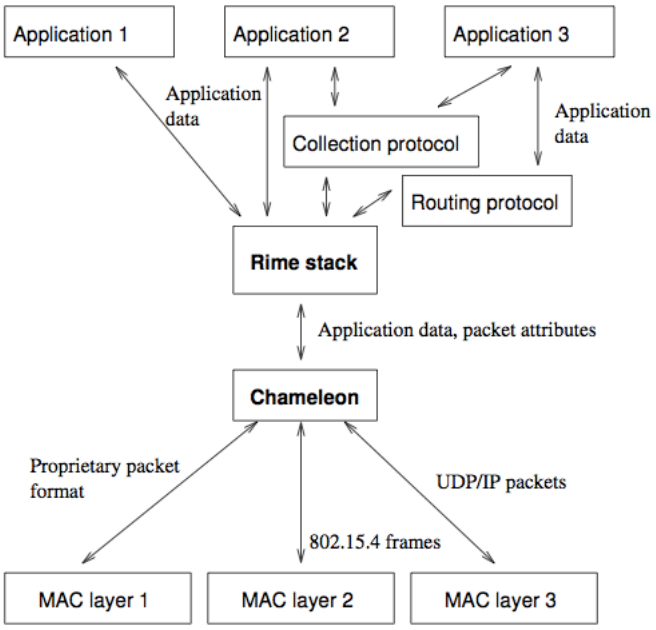


Figure 2.5: The Chameleon architecture. Applications and network protocols run on top of the Rime stack. The output from Rime is transformed into different underlying protocols by header transformation modules.

ies [Li et al., 2004] were carried out using ad-hoc network simulators, such as NS-2 [Simulator, 2010] or GlomoSim [Zeng et al., 1998]. Like TOSSIM [Levis et al., 2003] for TinyOS, Contiki OS offers a variety of simulation environments to evaluate systems performance: MSPsim [Eriksson et al., 2007], Netsim [Loubser, 2006] and COOJA [Österlind, 2006]. In fact, due to the distributed nature of sensor networks and resource-constraints of sensor nodes, code development for wireless sensor network is a challenging and time consuming task. Furthermore, the application development and debugging tools are still cumbersome. Therefore, the use of a simulator enables testing without access to the target hardware and allows more advanced debugging and instrumenting possibilities.

The Java-based MSP430 MSPsim has been developed as a tool for debugging and testing Contiki OS-based MSP430-based sensor network applications. MSPsim is an extensible sensor board platform and MSP430 instruction-level simulator that simulates sensor boards with peripherals (i.e. sensors, communication ports, LEDs, and sound devices such as a beeper) for the purpose of reducing development and debugging time. MSPsim can be, by design, incorporated within COOJAs cross-level simulation environment. Current versions have been developed with emulation of Scatterweb ESB and TMote Sky sensor nodes.

The Contiki OS Network Simulator, Contiki Netsim, is a simulator for sensor nodes running the Contiki OS. In Netsim, each sensor node is simulated within a graphical environment. Each node is represented as user-level process and it communicates with other nodes using a simulated wireless network layer. Although more advanced than MSPsim, Netsim is still a quite primitive and basic simulator: if message collision and interference occur, they are not detected and the information will never be able

to reach the destination. The lack of implementation of MAC protocols within the simulation environment is responsible for an unrealistic and implausible way to model radio communication and therefore applications.

The most sophisticated and modular Contiki OS simulator is COOJA. COOJA's main motivation concerns extensibility, and thus the possibility to include additional functionalities and models to the basic simulation environment through interfaces and plugins. The interface represents a sensor node property (e.g. location etc.) or a hardware device (e.g. button, radio transmitter, etc.). The plugin is used to interact with the simulation environment (e.g. control the simulation speed, monitor the network traffic between the simulated nodes, etc.). Additional plugins and interfaces are created and easily linked to the main simulation environment. Radio communication can be modelled, implemented and added to the simulator. The simultaneous support of several plugins within the same simulation environment enhance heterogeneous network simulation. Java Native Interface (JNI) is used to link the COOJA simulator with the underlying Contiki OS, allowing simulated applications to run the Contiki OS. Therefore, any application simulated within COOJA should be able to run unchanged on a real sensor node. Several plugins have been developed in order to deal with WSNs radio communications (e.g. Unit Disk Graph Medium (UDGM), Multi-path Ray-tracer Medium (MRM) [RadioCooja, 2010]). However, despite the considerable efforts made by simulators to model network traffic, they are still unable to provide a realistic snapshot of the complex and dynamic reality of radio communications.

The most common assumptions regarding radio communication models done within simulation environments were summarised in [Kotz et al., 2004, Ali et al., 2006] as follows: (i) the world is assumed to be flat; (ii) the radio transmission area of a node is assumed to be circular; (iii) radio communications are assumed to have equal transmission range; (iv) nodes are assumed to communicate with each other bidirectionally (i.e. if node A hears node B, then B equally hears A); (v) if node A hears node B, then perfect communication is assumed (i.e. A perfectly hears B); (vi) signal strength is assumed to be a simplified function of the distance among nodes. Instead, in the literature (e.g. [Kotz et al., 2004, Aguayo et al., 2004, Park et al., 2003, Kotz et al., 2003, Zhao and Govindan, 2003, Cerpa et al., 2003, Ganesan et al., 2002]) it is widely accepted that: (i) radio propagation is non-isotropic (i.e. the received signal, at a given distance from the sender, is not the same in all directions); (ii) it has non-monotonic distance decay (i.e. lower distance does not mean better link quality); and (iii) the communication is based on asymmetrical links (i.e. if A hears B, it cannot be assumed that B hears A).

Consequently, in meeting our research hypothesis, and thus measuring the impact of network conditions on the system performance whenever distributed processing becomes necessary, despite the presence of simulation environments, we decided to perform the whole experimental evaluation presented in the current dissertation entirely on deployed WSNs testbeds.

What is missing: Solutions supporting evaluations undertaken within deployed systems and testbeds.

2.3 Discussion

In this chapter, we identified the requirements common to the application scenarios presented in Section 1.1, thus (i) distributed processing, (ii) network communication, and (iii) evaluation platforms in WSNs. We then provided an extensive critical analysis of the existing related work undertaken to deal with such challenges. The weaknesses emerging from the related work analysis motivate the choices made within the current dissertation, and thus the need to devise a distributed solution: (i) allowing individual nodes to collaborate with each other in an ad-hoc manner to solve locally computationally demanding applications; (ii) allowing individual nodes to account for local network conditions in reasoning about task allocation; and (iii) supporting evaluations undertaken within deployed systems and testbeds.

Chapter 3

Distributed Wireless Ad-hoc Grids

In Chapter 2, we presented a set of scenarios and we listed their common issues and requirements. This chapter details, instead, the emergency scenario chosen as driving one within the current dissertation. Moreover, it presents the *Distributed Wireless Ad-hoc Grids* (DWAGs) paradigm together with the requirements for its practical applicability and the related challenges. A critical analysis of the related work places DWAG within the literature.

The remainder of this chapter is thus organised as follows: (i) in Section 3.1, we describe the *Distributed Wireless Ad-hoc Grids* paradigm; (ii) in Section 3.2, we analyse requirements of applications to allow DWAG applicability; (iii) in Section 3.3, we describe the main DWAG architecture and system flow; and (iv) in Section 3.4, we place DWAG within the existing related work.

3.1 Distributed Wireless Ad-hoc Grids Paradigm

We are now in a position to focus on the scenario adopted within the current dissertation and to present the proposed *Distributed Wireless Ad-hoc Grids* (DWAGs) paradigm.

Assume that we have an emergency scenario like that described in Section 1.1. Whenever a Chemical, Biological, Radiological or Explosive (CBRE) emergency event occurs within an indoor environment, it is crucial (i) to constantly monitor the hazardous environment (e.g. to detect the epicentre of the threat, its speed of spreading, its level of lethality), (ii) to progressively locate first responders while they move (e.g. whenever GPS support cannot be assumed), (iii) to guide personnel movements towards places at which injured people might be located, (iv) to support first responders in real-time finding evacuation routes according to how the emergency situation evolves over time. These are just a few examples of the applications required to be in place whenever an emergency event occurs. Pervasively deployed WSNs could support external *Command & Control* C^2 authorities in fetching such information and thus inferring reactive countermeasures.

In particular, let us thus assume a typical tactical WSN scenario as illustrated in Figure 3.1. In such a context, we have that a group of first responders enters an indoor environment (e.g. building, train station, underground, etc.) as a result of an emergency event (e.g. earthquake, fire, etc.). The environment might be subdivided into rooms and corridors. In this scenario, we assume that a large number of sensor nodes are distributed throughout a given space: some might be pre-deployed, some

might dynamically enter the environment (e.g. sensors worn by operational personnel, carried on UAVs, on robots, on animals, etc.). Let us name the pre-deployed devices *Task Executors* (TEs) (e.g. $TE_1, \ldots,$ TE_{12} in Figure 3.1), and the dynamically deployed ones *Grid Activators* (GAs) (e.g. $GA_1, \ldots, GA_5$ in Figure 3.1). Moreover, while pre-deployed TE nodes might be both constantly powered through the fixed power network and pre-loaded/periodically updated with a set of libraries, the mobile GAs might be battery-powered and dynamically programmed. The main task of a GA is the execution of a set of intensive computations, that can occasionally require more resources than those available to them. In particular, let us assume that there are GA_x with $x = \{1, \ldots, X\}$ and TE_y with $y = \{1, \ldots, Y\}$ in the environment and that each GA_x is required to compute one or more intensive applications Job_i with $i = \{1, \ldots, N\}$.

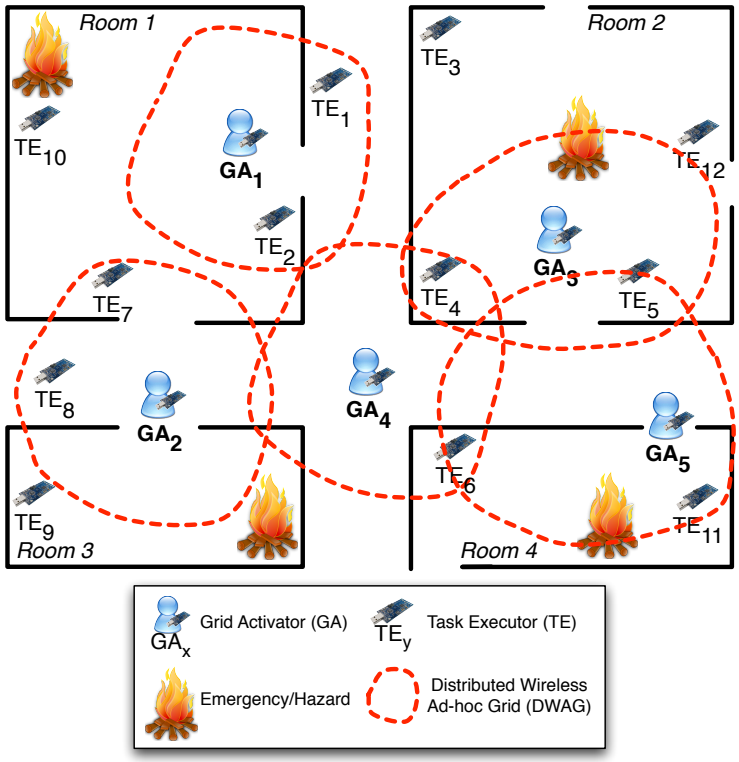


Figure 3.1: Example of an emergency scenario involving the *Distributed Wireless Ad-hoc Grids* paradigm. Let us assume the presence of a fire spreading within an indoor environment. In such a situation, the GAs are nodes lacking sufficient resources to perform a set of computationally demanding applications (e.g. node localisation, data compression, etc.), while the TEs are auxiliary nodes supporting the GAs in performing such computations. The core idea behind the node-centric DWAG is, in fact, the possibility for the GAs to fulfil their duties by taking advantage of the TEs by forming *dynamic ad-hoc grids* with them. By doing so, the GAs can thus rely upon computational resources ideally bigger than those locally available.

An example of an application needed in this situation is represented by node localisation. Since, in indoor environments, Global Positioning System (GPS) support cannot be assumed, the GA nodes would constantly need to autonomously compute their own location and to provide such summarised information to the external *Command & Control* (C^2) in order for it to keep real-time track of their movements, guide and aid them to fulfil their duties (i.e. locate and save injured people trapped inside,

extinguish fires, etc.). Therefore, although node localisation is only an example of the set of computationally demanding applications necessary to be in place whenever an emergency occurs, it is indeed crucial for such scenarios. For this reason, localisation has been chosen as main case-study for the current dissertation, and it is hence adopted to describe the envisioned DWAGs paradigm.

In order to perform the computation of intensive applications, such as for example real-time node localisation, each GA_x may choose either to individually and centrally execute job Job_i, or to distribute some of the tasks $Task_{i,j}$, of which the job is comprised, to some of its TE_y neighbours. At present, there are only two choices about where computation might occur within a network of devices: (i) on individual nodes, with the advantage of achieving substantial data reduction, decreasing the cost of transmission and avoiding congestion; (ii) outside the network, with the nodes simply supplying the data required for the calculation. The latter approach does remove the need for powerful processors on nodes, but necessitates a higher bandwidth network, whereas the tradeoffs are reversed for the former case. A simple solution to the problem would be to over-engineer both the computational capacity of a node and the network for data transmission; however, constraints of size and energy consumption may make this impracticable, particularly for mobile nodes. Moreover, the sophistication of applications, particularly if dynamically deployed, may change over the deployment lifetime of the devices that should be required to compute them in the first place. Therefore, regardless of a device's absolute computational power or network bandwidth, applications may reach a point at which they cannot be individually executed on a single node.

For this reason, we devise a distributed paradigm, namely DWAGs that, instead of relying upon centralised approaches, represents a distributed solution to solve computationally demanding applications by applying computational grid principles, as described in Section 3.4.1, to WSNs. The core idea of the node-centric DWAG paradigm is, in fact, the possibility for the GA nodes to fulfil their duties by forming in the environment *dynamic ad-hoc grids*, whenever needed, with the auxiliary TE nodes that are physically nearby (single-hop away). A requirement to consider in order for such distribution to take place is the expectation of continued network connectivity over the duration of each DWAG interaction. However since, during emergencies, applications are required to be computed as fast as possible (i.e. few seconds/minutes), we do not expect network connectivity to represent a limiting constraint for the considered scenario. Moreover, DWAG formations are lightweight and flexible, so that a GA can keep dynamically adding new TEs and removing those outside its communication range. Independent DWAG formations centred on different GAs can also overlap and use part of the same TEs. As illustrated in Figure 3.1, the GAs keep forming dynamic ad-hoc grids while moving, thus offloading computation on auxiliary nodes and asynchronously receiving back from them results.

Resource-intensive applications Job_i thus need to be split into smaller independent tasks $Task_{i,j}$ that are then offloaded from the GAs to the TEs to create node-centric DWAGs. In order for this to occur, schedulers need to be implemented on the GAs in order for them to make the best decision about the TEs on which the task offload should be performed. Once results for each distributed task $Task_{i,j}$ have been computed and sent back to the GA, a global result deriving from such individual computations

needs to be combined on the GA. The final aggregated information thus represent a summarisation of the data from which it was calculated, requiring reduced resources (e.g. bandwidth) to transmit throughout towards C^2 than would the raw information.

3.2 Application Requirements for DWAG Applicability

In order to allow computationally intensive applications to be executed by adopting the DWAG paradigm, they need to fulfil a set of requirements that are listed as follows:

1. Whenever an application Job_i needs to be computed, it must be possible to identify a set of *independent* $Task_{i,j}$, forming Job_i, that can thus be executed in *parallel*. Task identification can be performed either by hand or automatically through tools able to identify the main tasks and profile them. As stated in Section 1.4, it is beyond the scope of this dissertation the design of general tools able to automatically extract tasks $Task_{i,j}$ from each complex applications Job_i. Therefore, for the applications we target within this dissertation, we assume that tasks are manually identified.

2. Whenever tasks $Task_{i,j}$ have been identified, the code for such tasks must be loaded onto the nodes in the WSN. This can happen in a number of ways: typically, code will be distributed, or migrated while tasks are executed, from the GA to the TE nodes on-the-fly. Clearly, this approach relies upon the existence of a supporting Operating System (OS) or middleware (i.e. [Costa et al., 2007]) capable of dynamically loading and linking code. Other approaches include pre-deployment of useful code fragments to nodes that are likely to be used as TEs, or the retrieval of such code from a repository. Since DWAG has been designed in the first place to be independent from the underlying loading code infrastructure and since a variety of related work (i.e. [Costa et al., 2007]) already exists in terms of middleware capable of on-the-fly loading and linking code, we confine ourselves to code pre-deployment within this dissertation.

3. Whenever $Task_{i,j}$ are distributed, it must be possible to estimate their resource usage. In particular, tasks need to be profiled. Such a task profile can be determined statically, as a result of a static analysis, or on the basis of dynamic profiles that are established and progressively maintained through task execution. In the former, the feature extraction statically occurs only once at compile-time and no further profile update occurs at run-time. In the latter, the feature extraction and characterisation dynamically occurs at run-time, thus progressive task profile refinements are performed each time updates are required during job execution. Tactical WSNs are not a general purpose computing resource and, although the superset from which extant tasks are drawn is expected to evolve over the long term, it will change relatively slowly. Moreover, tasks in WSNs are frequently rather repetitive data analytic tasks. Thus, the conditions for both static analysis (i.e. slowly changing and restricted tasks amenable to pre-deployment code analysis and simulation) and dynamic analysis (i.e. repetitive tasks) hold. However, as stated in Section 1.4, the design of an automatic profiler is beyond the scope of this dissertation, therefore our system assumes the presence of an external profiler able both to identify $Task_{i,j}$ into which Job_i has been split and to characterise them in terms of the resources required in order for them to be computed.

3.3 DWAG System Flow Description

In order for demanding applications Job_i to be performed through the DWAG paradigm, a set of job requirements need to be taken into account. As described in Section 3.2, it must thus be possible: (i) to recognise the set of independent $Task_{i,j}$ into which Job_i can be split; (ii) to pre-load the TEs with a set of functionalities allowing them to support the GAs during the computation of the main application flow; (iii) to rely on a profile description for each $Task_{i,j}$, thus a set of relevant features $\{f_1, \ldots, f_F\}$ required to be extracted and used to characterise $Task_{i,j}$.

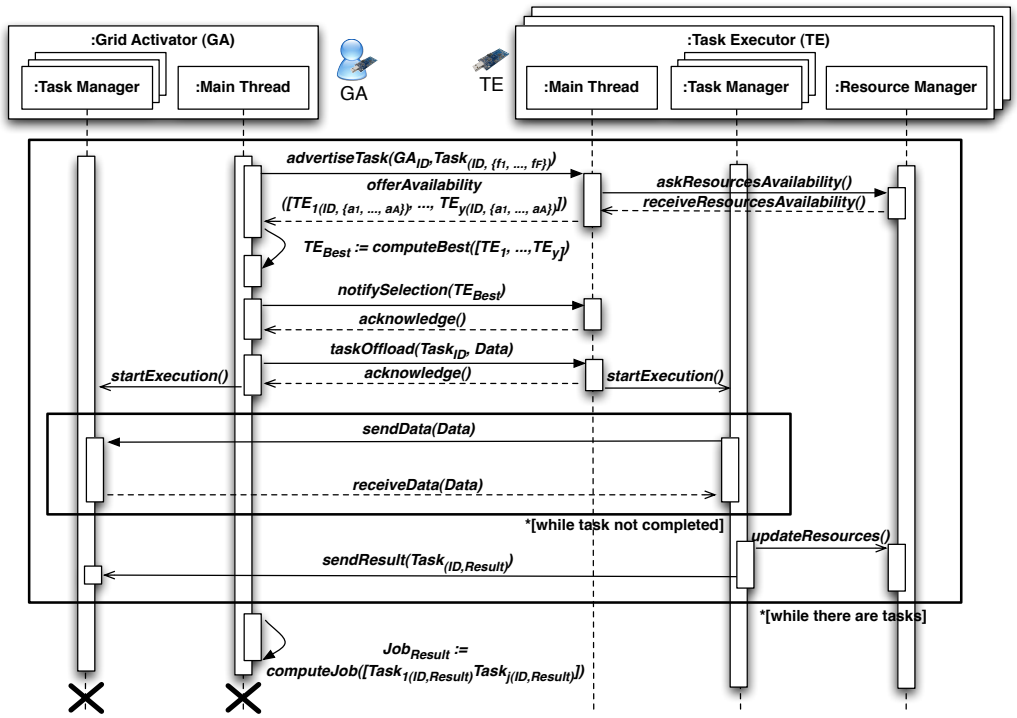


Figure 3.2: Main system flow and sequence of actions undertaken by the GA and the TE nodes whenever the DWAG paradigm is performed.

We are now in a position to describe the sequence of actions performed whenever a demanding application Job_i is executed through the DWAG paradigm. Recall that, although DWAG is applied to WSNs within the current dissertation, the paradigm itself is both technology and application-independent. The following represents a high-level, application-independent description of DWAG main system flow.

As illustrated in Figure 3.2, each time a node-centric dynamic ad-hoc grid is formed, a single GA and a multiplicity of TEs are involved. Moreover, the GA and the TEs are internally composed of a variety of processes, namely *Main Thread*, *Task Manger* and *Resource Manager*.

The GA has a unique *Main Thread* representing the main computational flow of the demanding application Job_i required to be collaboratively executed. This flow is interrupted by the presence of a multiplicity of tasks $Task_{i,j}$, into which Job_i has been split. Each $Task_{i,j}$ is externally profiled so that, when ready to be offloaded, it shows a set of characteristic features $\{f_1, \ldots, f_F\}$. For both the GA and the TE, each *Task Manager* instance takes care of an individual task offload. Thus, the number of *Task*

Manager instances concurrently active on the same node represents the degree of parallel computations it supports. The responsibilities of each *Task Manager* are twofold: (i) assisting the initial load distribution; (ii) supporting the remote task computation (e.g. sending data required to perform remote task execution, assisting message bursts, etc.).

Each TE is formed by the following components: (i) a single, always active, *Main Thread* instance ready to react to any request of task execution performed by the GA; (ii) a set of *Task Manager* instances each of which is responsible for the execution of a single task at the time; (iii) a single, always active, *Resource Manager* instance responsible for fetching information related to both the TE real-time internal resources (e.g. in terms of available computation) and real-time local network conditions (e.g. in terms of network communication).

We are now ready to describe the sequence of activities illustrated in Figure 3.2 and undertaken by the GA and the TE nodes each time DWAG applies. Whenever task $Task_{i,j}$ needs to be remotely offloaded, firstly the GA advertises it within the environment (through *advertiseTask*$(GA_{ID}, Task_{(ID,\{f1,\ldots,f_F\})})$ by stating its own node identifier GA$_{ID}$, the identifier of the task $Task_{ID}$, and the set of characterising features $Task_{\{f1,\ldots,f_F\}}$. The aim of this first phase is to dynamically and locally discover the GA's neighbourhood, and thus in-range TE$_{y_{ID}}$ together with their resource availabilities TE$_{y_{\{a1,\ldots,a_A\}}}$. Each one of the in-reach available nodes replies in fact to such request with an availability offer (through *offerAvailability*$([TE_{1_{(ID,\{a1,\ldots,a_A\})}}, \ldots, TE_{y_{(ID,\{a1,\ldots,a_A\})}}])$. On the TE$_y$ side, such information is fetched by querying (through *askResourcesAvailability()* and *receiveResourcesAvailability()*) the *Resource Manager* component that, being an always active process, constantly keeps real-time monitoring and updating the TEs resources and network information. The output of this phase is thus a list, representing the discovered *dynamic ad-hoc grid* neighbourhood, filled with the unique identifiers (e.g. unique IP addresses) of the neighbouring nodes and their respective resources availabilities. The requests are synchronously handled, thus no further action is undertaken by the GA unless the TEs are in range and provide their availability.

The GA needs now to select the best TE$_{Best}$ among those TE$_y$ belonging to the discovered neighbourhood (through TE_{Best} :=*computeBest*$([TE_1, \ldots, TE_y])$). After the choice has been made, the GA synchronously notifies the selection (through *notifySelection*(TE_{Best})) to the whole neighbourhood in order both to inform the TE$_{Best}$ to get ready to handle the offload and to unlock the other TE$_y$ still waiting to be potentially chosen. TE$_y$'s acknowledgments allow the GA to progress the task offload.

Thus, the TE$_{Best}$ receives the initial amount of data required to handle the remote computation of the offloaded $Task_{ID}$ (through *notifySelection*(TE_{Best})). From this moment on, the whole management of the task $Task_{ID}$ execution is delegated to the *Task Manager* components of both the GA and the TE$_{Best}$ (through *startExecution()*). In fact, any further request of data required to maintain the remote execution is directly managed by the GA and the TE$_{Best}$'s *Task Manager* components (through *sendData(Data)* and *receiveData(Data)*) until task execution is completed. At the end of the phase of progressive task computation, the TE$_{Best}$'s *Task Manager* keeps providing updates to the *Resource Manager* (through *updateResources()*) on the actual status of the TE$_{Best}$'s available resources. This is

crucial, since each TE is required to handle multiple requests at the same time. During the offer phase, the TE should be able promptly to provide as much up-to-date information as possible regarding the current status of its resources.

Once computed, the task result is thus directly sent back to the GA (through *sendResult*($Task_{(ID,Result)}$)). The entire sequence of actions is repeated for each task $Task_{i,j}$ into which Job_i has been split. Finally, since a result is generated for each computed task, they all need to be combined in order to create a unique Job_i result (through Job_{Result} :=*computeJob*($[Task_{1_{(ID,Result)}}, \ldots, Task_{j_{(ID,Result)}}]$)). Once the final job result has been computed, both the GA's main flow *Main Thread* and the *Task Manger* terminate their execution.

3.4 Related Work

We are now interested in discussing the position of the devised DWAG paradigm within the related work. In particular, we structure the discussion into two main parts, thus (i) we enhance DWAG features and we relate them according to the Grid Computing research field that inspired them; and (ii) we present existing approaches in the related work done to integrate Grid Computing and WSNs.

3.4.1 Grid Computing and DWAG Features

In this chapter, we presented the DWAG paradigm. This was proposed with the idea of applying some principles deriving from the Grid Computing field to networks of resource-constrained WSNs devices. We now provide a brief overview of computational grid systems, comparing them against clustering computing systems, with the aim of highlighting and characterising DWAG properties.

Grid Computing [Foster and Kesselman, 1997, Foster and Kesselman, 1999, Foster et al., 2001] (i.e. computational grid systems, grids) is a distributed computing paradigm referring to the ability of a federation of common computers connected to a network (e.g. private, public, Internet, etc.) by a conventional network interface (i.e. ethernet) to collaborate with each other to form a logical Virtual Organisation (VO) with the aim of gathering and sharing computing resources, similarly to a multi-processor super-computer, and then distributing process execution across a parallel infrastructure. Grid middleware (i.e. Globus Toolkit [Foster and Kesselman, 1997], gLite [gLite, 2010], and UNICORE [UNICORE, 2010]) is a specific software product, representing a special layer placed between the heterogeneous infrastructure and the specific user applications, enabling the sharing of heterogeneous resources among Virtual Organisations.

Grid Computing systems greatly differ from the traditional notion of single super-computer. In traditional super-computers, in fact, several powerful processors are embedded within the same hardware and connected with each other through a local high-speed computer bus. On the contrary, each node in a computational grid system can be purchased as commodity hardware which, when combined with other nodes, can logically emulate computing resources similar to those of a multi-processor super-computer, but at lower cost. Hence, Grid Computing provides the ability to achieve higher computing throughput by taking advantage of the unused resources of a federation of heterogeneous computational nodes modelling a virtual computer architecture. Nodes belonging to a grid are then able to individually solve

	Grid Computing	**Cluster Computing**
System nodes	Loosely-coupled	Tightly-coupled
Geographic distribution	Dispersed	Centralised
Infrastructure	Distributed	Fixed
Node hardware	Heterogeneous	Homogeneous
Job scheduling/management	Distributed	Centralised

Table 3.1: Grid computing vs. cluster computing systems.

smaller tasks, into which a computationally intensive application has been split, by distributing process execution across an infrastructure in which multiple parallel computations take place, independently. The grid infrastructure manages the logic behind the allocation process so that, once distributed, single tasks are executed independently and intermediate task results do not affect other in-progress task executions. Therefore, computational grid systems are composed of heterogeneous loosely-coupled computers, geographically dispersed but connected through a high-speed network, orchestrating with each other to solve large-scale computational problems requiring a great number of computer processing cycles or the access to large amounts of data.

To date, Grid Computing has also been applied to computationally intensive scientific, mathematical, and academic problems through volunteer computing, and it is used in commercial enterprises for diverse applications. For example, some of the most popular Grid Computing systems are the following: search for extraterrestrial intelligence in SETI@Home [SETI@Home, 2010]; protein folding in Folding@Home [Folding@Home, 2010]; physics research like developing and exploiting particle accelerators such as CERN Large Hadron Collider LHC@Home [LHC@Home, 2010]; dealing with humanitarian causes in Africa@Home [Africa@Home, 2010]; earthquake simulation, climate/weather modelling, linked environments for atmospheric discovery in LEAD [LEAD, 2010]. A number of corporations, professional groups, university consortiums, and other groups have developed frameworks and software for managing computational grid projects. The European Union (EU), National Technology Grid, Sun Microsystems, together with several other associations, are the main supporters of Grid Computing.

Grid Computing is a specialisation of cluster computing [Buyya, 1999a, Buyya, 1999b]. However, cluster computing greatly differs from computational grids for the following reasons. Firstly, cluster computing nodes are tightly-coupled among each other, thus remote task computations are not executed autonomously but there is, instead, strong control coming from head nodes (e.g. cluster-heads) in the architecture. Moreover, nodes are often organised according to a static pre-defined structure (e.g. hierarchical). In addition, they are mainly homogeneous and geographically centralised. Finally, job management and scheduling is not distributed on each node but it is managed by a centralised authority. Grid Computing and clustering computing features are compared in Table 3.1.

The devised DWAG paradigm represents a tentative to port Grid Computing features within networks of resource-constrained devices, such as WSNs. Therefore, Grid Computing characteristics have been adapted within the DWAG paradigm, as follows:

- *Loosely-coupled nodes:* Whenever each task, into which a computationally intensive job has been split, is distributed from the GAs to the auxiliary TEs nodes, task execution is performed in a completely autonomous way on the TEs. Although data offload and upload are performed, DWAG underlying node infrastructure is completely decentralised, thus grids are ad-hoc created, and the TEs autonomously perform task computation.

- *Geographically dispersed nodes:* In DWAG, nodes are exclusively required to be wirelessly in reach with each other to be able to communicate, but there is no constraint concerning their geographical placement in the environment.

- *Distributed infrastructure:* the GAs dynamically create distributed wireless ad-hoc grids, relying upon a completely dynamic infrastructure in which nodes add or remove themselves according to their available computational capabilities.

- *Heterogeneous nodes:* In DWAG, as long as nodes interact with each other through the same communication protocol, no requirements of hardware homogeneity is assumed.

- *Distributed job scheduling/management:* Each GA node initiating DWAG must deal with task scheduling and management algorithms, since task distribution is completely decentralised and managed by single nodes, individually.

Finally, note that, like the Globus Toolkit for Grid Computing, DWAG relies upon job characterisation and knowledge of the resources required to be allocated during the task distribution process. Although Globus Toolkit represents a flexible and modular tool, its implementation is not lightweight enough to run on networks of resource-constrained devices.

3.4.2 DWAG vs. Sensor Grid Systems

In the past few years, an increasing trend towards the integration of WSNs and Grid Computing fields has been recorded [Coulson et al., 2006], leading to the development of a great variety of Sensor Grid systems.

Sensor Grids represent the tentative of combining WSNs and Grid Computing features to generate a unifying system. As described in Section 2.2.1, to date, WSNs have been adopted with the idea of instrumenting the physical environment with resource-constrained, low cost devices able to monitor environmental phenomena by progressively gathering data readings. As described in Section 3.4.1, Grid Computing is instead a distributed computing paradigm allowing nodes, required to perform intensive computations, to exploit the unused resources of a federation of additional heterogeneous nodes, distributed across several administrative domains and modelling a virtual computer architecture, to achieve job computation. Sensor Grid systems combine the two paradigms through infrastructures in which real-time environmental data are: (i) collected by heterogeneous and distributed sensing nodes; and (ii) real-time analysed by exploiting the computational resources of a pool of machines (e.g. computers) belonging to several virtual organisations. Sensor Grid is therefore a technology for building large-scale infrastructures integrating heterogeneous sensors, data and computational resources deployed over

a wide area, to undertake computationally intensive tasks.

Sensor Grids have several motivations: firstly, the vast amount of data collected by sensors can be processed, analysed, and stored using the computational and data storage resources of a Grid Computing infrastructure; secondly, sensors can be efficiently shared by different users and applications under flexible usage scenarios. Hence each user can access a subset of nodes for a certain time period to run specific applications and to collect the desired type of data. Finally, Sensor Grid allows the pervasive access to a wide variety of resources. Data fusion, mining and distributed database processing techniques can thus be applied to generate environmental knowledge, improve sensor collection and influence environmental actuators.

Several pervasive support systems (e.g. GridStix [Hughes et al., 2006], SPRING [Lim et al., 2005], Code Blue [CodeBlue, 2010], Discovery Net [Net, 2010], etc.) have been developed to predict and manage environmental threats (e.g. fire, flooding [Coulson, 2006, Hughes et al., 2006], etc.), to support emergency rescue services [Lorincz et al., 2004], to monitor weather [Lim et al., 2007], to react to strains in engineering artefact (e.g. bridges, aircraft engines, etc.) and for healthcare purposes [Oh and Lee, 2008]. Therefore, the infrastructures [Lim et al., 2005, Tham and Buyya, 2005, Gaynor et al., 2004] enable the construction of real-time models and databases of the environment and physical processes as they unfold, from which high-value computations like decision-making, analytics, data mining, optimisation and prediction can be carried out to generate on-the-fly results. A survey [Ahuja and Myers, 2006] for Sensor Grid infrastructures has been provided. Since WSNs are exposed to fixed and long-term deployments, the main concern is related to power efficiency and battery lifetime maximisation.

Our DWAG paradigm greatly differs from the rationale behind Sensor Grid systems. Firstly, sensor devices are employed in DWAGs not with a *passive*, but with an *active* role, since they are adopted as entities that, because of their computational capabilities, are able to collaboratively perform more complex and intensive applications. Secondly, in DWAG the grid is entirely initiated and built among sensor platforms themselves without relying upon extra computational capacity deriving from additional high-performing computing nodes: sensor devices are not themselves seen as mere data collectors, but actual executing nodes. Finally, the DWAG infrastructure is completely flexible and distributed, thus each node is able, according to its needs, to initiate the grid or perform the functionalities of auxiliary nodes.

3.5 Discussion

In this chapter, we presented the Distributed Wireless Ad-hoc Grids paradigm employed to collaboratively execute computationally intensive applications by locally distributing tasks, into which a main job is split, within the network. We listed the application requirements to allow DWAG applicability. We then described the sequence of actions to be undertaken in order to perform DWAG. Finally, we located the paradigm within the related work emphasising analogies and differences with traditional distributed computing approaches like Grid Computing and cluster computing.

Chapter 4

Impact of Network Conditions on DWAG in Deployed Systems

In Chapter 3, we described the scenario and the DWAG paradigm together with the requirements for its practical applicability and the associated related challenges. In particular, the major weaknesses of the related work presented in Chapters 2 and 3 are summarised as follows: (i) most of the approaches exclusively focus on increasing algorithmic sophistication, largely ignoring practical issues pertaining to real-world radio communications; (ii) most of the evaluations are performed through simulations within simulation environments making implicit assumptions about persistent radio medium reliability. Although these assumptions are convenient for simulation purposes, they happen to be inadequate when it comes to evaluating algorithms in deployed testbeds. The objective of this chapter is thus to investigate the impact of environmental network traffic contention on the distribution, according to the DWAG paradigm, of computationally intensive applications within deployed systems.

The remainder of this chapter is thus organised as follows: (i) in Section 4.1, we describe the bandwidth problem as the impact of environmental network traffic contention on the distribution of computationally intensive applications according to the DWAG paradigm; (ii) in Section 4.2, we present two load sharing algorithms, namely the Auction and the Lookup List, chosen and adapted to be implemented in a completely distributed manner on deployed testbeds with the aim of load distribute applications; (iii) in Section 4.3, we devise and integrate within the algorithmic decision making process of the basic load distribution algorithms a Bandwidth-Aware Task Scheduling (BATS) heuristic mechanism combining the local information regarding the TEs available computational resources and the local amount of existing network contention in the areas where the TEs are located; (iv) in Section 4.5, we briefly discuss related work on load distribution; (v) in Section 4.6, we test the proposed DWAG paradigm with related bandwidth-aware heuristic with actual computation, actual profiling of the medium, and actual network traffic for a multiplicity of settings and we assess the impact of network conditions on distribution of homogeneous tasks within deployed systems; and (vi) in Section 4.7, we summarise the obtained results.

4.1 Network Contention Impact on Distribution

As illustrated in Section 3.3, whenever collaborative computation takes place through the DWAG paradigm, the set of tasks $Task_{i,j}$, of which Job_i is comprised, are required to be distributed within the environment from the GA to the auxiliary TE nodes. In particular, before being distributed, a task $Task_{i,j}$ is assumed to be externally profiled and thus a set of characterising features $\{f_1, \ldots, f_F\}$ are extracted and further used as task descriptor. Similarly, before becoming the remote auxiliary node responsible for computing $Task_{i,j}$, the TE needs to advertise its resources availabilities $\{a1, \ldots, a_A\}$ to the GA. As described in Section 2.2.1, since WSNs are comprised of unsophisticated resource-constrained devices, both task profiling features and nodes resources availabilities are characterised through their computational (e.g. CPU cycles/active processes) and communication (e.g. bandwidth occupation/availability) requirements.

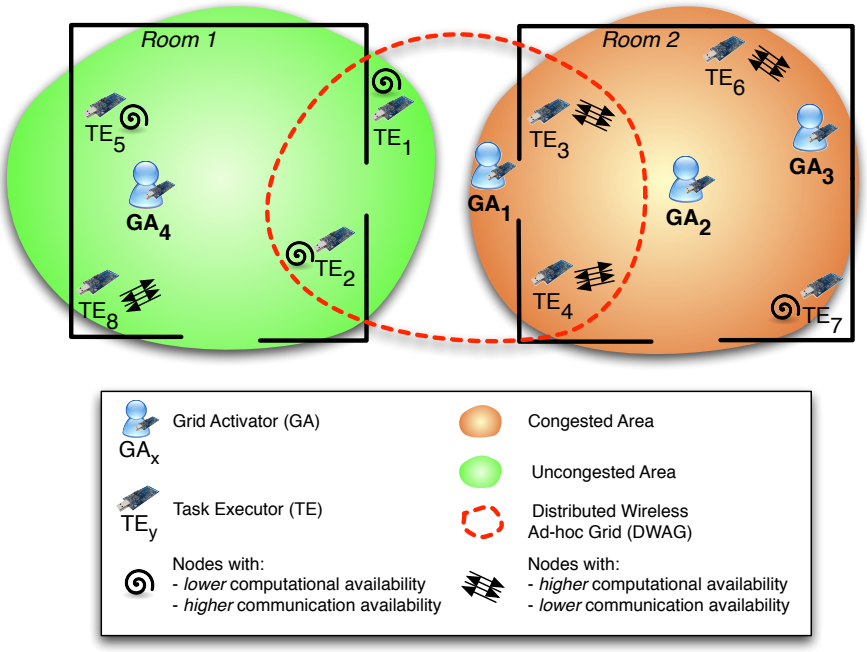


Figure 4.1: Network contention impact on distribution through DWAG. Several GAs keep distributing $Task_{i,j}$, of which their Job_i are comprised, to the TEs by building dynamic ad-hoc grids in the environment. For example, GA_1 needs to choose the best TE_y among those belonging to the dynamically discovered $\{TE_1, \ldots, TE_4\}$ neighbourhood. In a scenario where TE_y are both unequally loaded (e.g. $\{TE_1, TE_2\}$ have lower computational and higher communication availabilities, $\{TE_3, TE_4\}$ have a reversed situation) and network utilisation is unevenly distributed (e.g. $\{TE_1, TE_2\}$ and $\{TE_3, TE_4\}$ are located in congestion-free and contended areas, respectively), it is crucial to account not only for the impact of computational conditions but also for those of network conditions, in terms of radio communication contention, on distribution performance.

In particular, let us assume that we have a situation like that illustrated in Figure 4.1 where several GA_x, required to execute a demanding Job_i, keep distributing their $Task_{i,j}$ to TE_y by building virtual ad-hoc grids in the environment. For example, let us now focus on GA_1's behaviour. Since GA_1's neighbourhood is composed of $\{TE_1, \ldots, TE_4\}$, GA_1 needs to select the best TE_y able to handle $Task_{i,j}$ computation in the most efficient and effective way, namely completing $Task_{i,j}$ execution in the least

amount of time. For each DWAG, the neighbourhood dynamically changes thus, for each task computation, the offload decision might involve a different set of nodes.

In each instant of time, TE_y advertises its real-time internal resource availabilities, since each node could potentially be busier than others either from a computational or from a communication perspective. In Figure 4.1, nodes $\{TE_1, TE_2\}$ have a lower computational availability, since they are already busy performing intensive computations, and a higher communication availability since for example tasks, that each of them are required to fulfil, do not involve several message exchanges (e.g. spiral representation for $\{TE_1, TE_2\}$ in Figure 4.1). The opposite situation occurs for nodes $\{TE_3, TE_4\}$, having instead higher computational and lower communication availabilities (e.g. representation through sequence of arrows for $\{TE_3, TE_4\}$ in Figure 4.1), respectively.

A crucial aspect that cannot be disregarded in such scenarios is the impact of the local environment where TE_y nodes happen to be located at distribution time. In situations affected by considerable amounts of radio communication, it is unrealistic to assume a homogeneous distribution of both network utilisation and interference in the environment. In Figure 4.1, nodes $\{TE_1, TE_2\}$ and $\{TE_3, TE_4\}$ might be located in congestion-free or contended areas, respectively (e.g. they are in areas with a higher density of nodes, or with many tasks to be performed requiring relatively high network utilisation, or with few tasks requiring massive bandwidth usage, or again areas traversed by routes for substantial amounts of cross traffic). Moreover, whenever distribution becomes necessary, there are several reasons why it might itself add to the congestion of an area. Primary amongst them are the initial distribution of data and/or code, any subsequent TE_y requirement to communicate with the originating GA, and the control overhead of maintaining accurate information about TE_y state at the GA side.

Traditional distributed systems (e.g. computational grids) perform distribution relying upon stable network infrastructures where hosts are permanently connected to the main network through high-bandwidth and high-quality links. In our case, most of the distribution occurring within WSNs or among embedded devices communicating with each other through radio medium, lacks of such reliable communication links and it is thus more prone to congestion, collision, and interference. Since network traffic plays a crucial role whenever distribution wirelessly takes place, in the current dissertation we build an empirical analysis investigating the impact of network conditions on distribution performance. In particular, we analyse if, and to what extent, combining real-time information on local network utilisation to computational requirements indeed leads to greater overall system performance (i.e. fast job completion) than the case in which the load is merely distributed accounting for computational requirements.

Consequently, we are now in a position to state that, among the set of both task feature $\{f_1, \ldots, f_F\}$ and TE availability $\{a1, \ldots, a_A\}$ characterisations, we focus our research interest on two dimensions: computation and communication. In particular, we will refer to $Task_{i,j(f_{CPU}, f_{Band})}$ and $\text{TE}_{y(a_{CPU}, a_{Band})}$ to indicate task features and TE availabilities, respectively.

4.2 Load Distribution Algorithms

In Section 3.3, we described the high-level, application-independent flow representing the sequence of actions undertaken whenever a demanding Job_i is executed through the DWAG paradigm. In such a

system design, the entire negotiation phase anticipating both the actual offload and the further remote computation is carried on with the aim of identifying, for each dynamically built DWAG, the best TE_y node to which $Task_{i,j}$ execution needs to be assigned.

Since this dissertation aims to address the research problem of analysing the impacts of real-time local network conditions during collaborative computation rather than the creation of novel load sharing protocols, we choose, modify, and adapt two existing load sharing algorithms to account for network utilisation and local traffic congestion. Recall that, since the whole evaluation was performed on significantly resource-constrained embedded TMote Sky devices, algorithmic simplicity has always been preferred to sophistication.

The first protocol, described in Section 4.2.1, is a GA-initiated load sharing algorithm while the second one, described in Section 4.2.2, is inspired by the dynamic TE-initiated load sharing algorithm presented in [Chuang and Cheng, 2002]. In the remainder of this dissertation, we will refer to the former as the *Auction* algorithm and to the latter as the *Lookup List* algorithm.

Algorithm 1: Auction Algorithm - Grid Activator

/* Job_i has been externally split into a set of tasks $Task_{i,j}$. */
/* Each $Task_{i,j}$ is assumed to be profiled $Task_{i,j(f_{CPU},f_{Band})}$. */
/* GA has a single *Main Thread* and multiple *Task Managers*. */
/* $\#TaskManager_{active}$ is the number of running tasks. */
/* $\#TaskManager_{available}$ is the degree of multi-threading. */
/* **Main Thread:** */
foreach *($Task_{i,j}$)* **do**
  wait until $\#TaskManager_{available} > 0$;
  repeat
    broadcast a request advertising $Task_{i,j(f_{CPU},f_{Band})}$;
    wait time Δt_1 to receive bids;
  until *(at least one bid from TE_y is received)* ;
  select TE_{Best} by computing the best bid;
  repeat
    broadcast a message notifying TE_{Best} selection;
    wait time Δt_2 to receive TE_{Best} ACK confirming selection notification;
  until *(ACK is received)* ;
  repeat
    offload $Task_{i,j}$ data;
    wait time Δt_3 to receive TE_{Best} ACK confirming data offload reception;
  until *(ACK is received)* ;
  $\#TaskManager_{available} - -$;
  $\#TaskManager_{active} + +$;
  launch a *Task Manager* instance;
end
wait until all $Task_{i,j}$ results are received from *Task Manager*;
compute Job_i result combining all $Task_{i,j}$ results;
/* **Task Manager:** */
repeat
  wait time Δt_4 to receive data required to support $Task_{i,j}$ computation;
  send data required to support $Task_{i,j}$ computation;
until *($Task_{i,j}$ has not reached completion)* ;
receive $Task_{i,j}$ result when completion has been achieved;
$\#TaskManager_{available} + +$;
$\#TaskManager_{active} - -$;

Algorithm 2: Auction Algorithm - Task Executor

```
   /* TE has a single Main Thread and multiple Task Manager.        */
   /* #TaskManager_active is the number of running tasks.           */
   /* #TaskManager_available is the degree of multi-threading.      */
   /* TE_y(a_CPU) = #TaskManager_available                          */
   /* Main Thread:                                                  */
 1 repeat
 2     receive advertising Task_{i,j(f_CPU,f_Band)} offload request;
 3     gather resource availabilities TE_{y(a_CPU,a_Band)};
 4     if (#TaskManager_available > 0) then
 5         send bid to GA containing its internal resource availability TE_{y(a_CPU,a_Band)};
 6     end
 7     if (TE_y == TE_Best) then
 8         send ACK confirming selection notification;
 9         wait time Δt_1 to receive Task_{i,j} data;
10         send ACK confirming data offload reception;
11         #TaskManager_available − −;
12         #TaskManager_active + +;
13         launch an instance of Task Manager;
14     end
15 until (TE_y is active) ;
   /* Task Manager:                                                 */
16 repeat
17     send data required to support Task_{i,j} computation;
18     wait time Δt_2 to receive data required to support Task_{i,j} computation;
19 until (Task_{i,j} has not reached completion) ;
20 send Task_{i,j} result when completion has been achieved;
21 #TaskManager_available + +;
22 #TaskManager_active − −;
23 update internal resource availabilities TE_{y(a_CPU,a_Band)};
```

4.2.1 The Auction Algorithm

The first protocol, namely the *Auction* algorithm, is a GA-initiated load sharing algorithm, in which state information exchange is reactively handled.

In order to achieve Job_i completion, the GA needs to distribute the set of tasks $Task_{i,j}$, of which Job_i is comprised, to the TEs. Each $Task_{i,j}$ is assumed to be externally profiled and thus, when ready to be offloaded, it shows a structure of characterising features $Task_{i,j(f_{CPU},f_{Band})}$ that can be explained as follows: (i) $Task_{i,j(f_{CPU})}$ identifies the number of CPU cycles required; (ii) $Task_{i,j(f_{Band})}$ identifies the burst of messages to be exchanged to achieve $Task_{i,j}$ completion.

As illustrated in Section 3.3, both the GA and the TE have a single instance of *Main Thread* and multiple instances of *Task Manager* components. The former manages the main application Job_i flow for the GA and negotiates remote $Task_{i,j}$ offload for the TE. The latter instead handles the actual task execution on both the GA and the TE. In particular, whenever a task enters the execution, a specific instance of *Task Manager* is reserved on both the GA and the TE. Thus, while *TaskManager*$_{active}$ identifies the real-time number of parallel tasks running on a node, *TaskManager*$_{available}$ identifies the actual degree of parallelism (i.e. multi-threading) handled by each node. Moreover, the TE has also a *Resource Manager* component, an always active process responsible to provide real-time information on

both node computational and communication resources. It is crucial to point out that, as described in Section 4.6, the resource-constrained TMote Sky devices, elected for the testing of our DWAG, do not have embedded hardware multi-threading features and thus the parallelism and distribution have been handled in software. Moreover, multiple requests are managed by each TE on a First-Come First-Served (FCFS) basis.

Since the algorithms are fully distributed among the two main system actors GA and TE, we are now in a position to describe the main algorithmic steps undertaken by both the GA and the TE while performing the Auction algorithm, thus in Algorithm 1 and 2, respectively. Recall that since the *Resource Manager* is an always active component whose unique responsibility is to provide real-time information on node computational and communication availabilities, for the sake of simplicity we will not introduce it as a separate entity within TE algorithmic explanation, but instead we will simply refer to it any time internal resource availability information is fetched within a TE node.

For each $Task_{i,j}$ required to be remotely computed, the following actions need to be undertaken. Firstly, the GA needs to wait until there is at least one *TaskManager*$_{available}$ available to handle a remote offload. Then, the GA's *Main Thread* broadcasts a task request message containing its own node identifier, the identifier of the task ($Task_{i,j}$), and advertising $Task_{i,j(f_{CPU},f_{Band})}$ computational and communication features. The GA thus waits Δt_1 time to receive bids from in-reach TE$_y$ in order to discover its own DWAG neighbourhood. If no bid is received, the GA's *Main Thread* keeps broadcasting the request message until a set of bids are received. Upon receiving $Task_{i,j(f_{CPU},f_{Band})}$ offload request, the TE checks with the *Resource Manager* its internal resource availabilities TE$_{y(a_{CPU},a_{Band})}$. If at least a *Task Manager* is available to handle the request, then the TE sends a bid to the GA containing its internal resource availabilities TE$_{y(a_{CPU},a_{Band})}$. In particular, TE$_{y(a_{CPU})}$ identifies the number of available *TaskManager*$_{available}$ in charge to take care of remote task computation, while TE$_{y(a_{Band})}$ identifies the local level of bandwidth availability and thus describes the level of network congestion of an area.

Once all bids have been collected, and thus the DWAG neighbourhood has been discovered, the GA's *Main Thread* performs a computation to determine the best bid and thus selects the TE$_{Best}$. In Section 4.3, we will detail how the decision making process is handled at the GA side.

Upon selecting the TE$_{Best}$, the GA's *Main Thread* broadcasts a message notifying TE$_{Best}$ selection. This action has two effects: (i) the chosen TE$_{Best}$ locks the resources advertised through the bid by participating in the auction; (ii) the other TE$_y$, involved in the auction but not being chosen, are informed about the winner of the auction, and thus they free their internal resources potentially ready to be allocated for remote task execution.

Then, the GA keeps waiting a time Δt_2 and broadcasting the message notifying the TE$_{Best}$ selection until it receives the ACK from the TE$_{Best}$ confirming the selection notification. Once the ACK is received, the GA offloads the preliminary data necessary to allow $Task_{i,j}$ computation at TE$_{Best}$ side. The GA waits a time Δt_3 and offloads data until it receives the ACK from TE$_{Best}$ confirming the data offload reception. Upon receiving the ACK, the GA launches an independent *Task Manager* instance responsible to handle the remote task execution. Moreover, it updates its resource availabilities

by increasing the number of *TaskManager*$_{active}$ and decreasing those available *TaskManager*$_{available}$. Analogously, after sending ACKs confirming both the node selection notification and the data offload reception, the TE$_{Best}$ activates a parallel instance of *Task Manager* and it locks the set of resources ready to be used. In particular, it increases the number of *TaskManager*$_{active}$ and decreases those available *TaskManager*$_{available}$.

In the negotiation phase, the messages exchanged are synchronously handled. This implies that no further action is undertaken within the protocol unless an ACK (either positive or negative) is received. If no ACK is received within a defined time interval Δt, the procedure iterates and a new message is delivered.

However, whenever *Task Manager* instances are instead launched at both the GA and the TE$_{Best}$'s sides, they are individually and fully responsible for carrying on $Task_{i,j}$'s execution. Until $Task_{i,j}$ has reached completion, the GA and the TE$_{Best}$'s *Task Managers* communicate with each other every time it becomes necessary to exchange data required to support $Task_{i,j}$ computation.

Upon achieving $Task_{i,j}$ completion, the TE$_{Best}$ asynchronously delivers the results to the GA and it further updates its internal resource availabilities TE$_{Best(a_{CPU},a_{Band})}$. In particular, the TE$_{Best}$ increases the number of available *TaskManager*$_{available}$, while decreasing the number of active *TaskManager*$_{active}$ ones. As soon as $Task_{i,j}$ results are received at the GA side, the GA similarly frees its internal resources, thus increasing the number of available *TaskManager*$_{available}$, while decreasing the number of active *TaskManager*$_{active}$ ones.

Whenever results for all tasks $Task_{i,j}$ of which Job_i is comprised have been computed, the GA computes the global Job_i result by combining all single task $Task_{i,j}$ results.

4.2.2 The Lookup List Algorithm

The second protocol, the *Lookup List* algorithm, is a more sophisticated TE-initiated load sharing algorithm, in which state information exchange is proactively handled. The main idea behind this algorithm comes from [Chuang and Cheng, 2002], but it has been adapted to allow for network control.

Analogously to the Auction algorithm, in order to achieve Job_i completion, the GA needs to distribute the set of tasks $Task_{i,j}$, of which Job_i is comprised, to TEs. Each $Task_{i,j}$ is assumed to be externally profiled and thus, when ready to be offloaded, it shows a structure of characterising features $Task_{i,j(f_{CPU},f_{Band})}$ that can be explained as follows: (i) $Task_{i,j(f_{CPU})}$ identifies the number of required CPU cycles; (ii) $Task_{i,j(f_{Band})}$ identifies the burst of messages to be exchanged to achieve $Task_{i,j}$ completion.

As illustrated in Section 3.3 and specified for the Auction algorithm in Section 4.2.1, both the GA and the TE have a single instance of *Main Thread* and multiple instances of *Task Manager* components. The former manages the main Job_i flow for the GA and negotiates remote $Task_{i,j}$ offload for the TE. The latter instead handles the actual task execution on both the GA and the TE. In particular, whenever a task is executed, a specific instance of *Task Manager* is reserved on both the GA and the TE. Thus, while #*TaskManager*$_{active}$ identifies the real-time number of parallel tasks running on a node, #*TaskManager*$_{available}$ identifies the actual degree of parallelism (i.e. multi-threading) handled by each

Algorithm 3: Lookup List Algorithm - Grid Activator

```
   /* Job_i has been externally split into a set of tasks Task_{i,j}.        */
   /* Each Task_{i,j} is assumed to be profiled Task_{i,j(f_CPU,f_Band)}.     */
   /* GA has a single Main Thread and multiple Task Managers.                */
   /* #TaskManager_active is the number of running tasks.                    */
   /* #TaskManager_available is the degree of multi-threading.               */
   /* Main Thread:                                                           */
 1 repeat
 2     broadcast a request advertising need for remote task offload;
 3     wait time Δt_1 to receive availabilities;
 4     fill a Lookup List with internal resource availabilities TE_{y(a_CPU,a_Band)};
 5 until (at least N availabilities from TE_y are received) ;
 6 foreach (Task_{i,j}) do
 7     wait until #TaskManager_available > 0;
 8     repeat
 9         select TE_Best by computing the best availability;
10         repeat
11             send a message to TE_Best notifying its selection;
12             wait time Δt_2 to receive TE_Best ACK confirming selection notification;
13         until (ACK is received) ;
14         receive and update Lookup List with internal resource availabilities
           TE_{Best(a_CPU,a_Band)};
15     until (ACK is positive) ;
16     repeat
17         offload Task_{i,j} data;
18         wait time Δt_3 to receive TE_Best ACK confirming data offload reception;
19     until (ACK is received) ;
20     #TaskManager_available − −;
21     #TaskManager_active + +;
22     launch a Task Manager instance;
23 end
24 wait until all Task_{i,j} results are received from Task Manager;
25 compute Job_i result combining all Task_{i,j} results;
   /* Task Manager:                                                          */
26 repeat
27     wait time Δt_4 to receive data required to support Task_{i,j} computation;
28     send data required to support Task_{i,j} computation;
29 until (Task_{i,j} has not reached completion) ;
30 receive Task_{i,j} result when completion has been achieved;
31 receive and update Lookup List with internal resource availabilities TE_{Best(a_CPU,a_Band)};
32 #TaskManager_available + +;
33 #TaskManager_active − −;
```

Algorithm 4: Lookup List Algorithm - Task Executor

```
   /* TE has a single Main Thread and multiple Task Manager.        */
   /* #TaskManager_active is the number of running tasks.           */
   /* #TaskManager_available is the degree of multi-threading.      */
   /* TE_y(a_CPU) = #TaskManager_available                          */
   /* Main Thread:                                                  */
 1 repeat
 2     receive offload request;
 3     gather and send resource availabilities TE_y(a_CPU,a_Band) to GA;
 4     if (TE_y == TE_Best) then
 5         send ACK confirming selection notification;
 6         send update with internal resource availabilities TE_Best(a_CPU,a_Band) to GA;
 7         wait time Δt_1 to receive Task_i,j data;
 8         send ACK confirming data offload reception;
 9         #TaskManager_available − −;
10         #TaskManager_active + + ;
11         launch a Task Manager instance;
12     end
13 until (TE_y is active) ;
   /* Task Manager:                                                 */
14 repeat
15     send data required to support Task_i,j computation;
16     wait time Δt_2 to receive data required to support Task_i,j computation;
17 until (Task_i,j has not reached completion) ;
18 send Task_i,j result when completion has been achieved;
19 send update with internal resource availabilities TE_Best(a_CPU,a_Band) to GA;
20 #TaskManager_available + +;
21 #TaskManager_active − −;
22 update internal resource availabilities TE_y(a_CPU,a_Band);
```

node. Moreover, the TE has also a *Resource Manager* component, an always active process responsible for providing real-time information on both node computational and communication resources. Multiple requests are managed by the TEs using a FCFS strategy.

Since the algorithms are fully distributed among the two main system actors GA and TE, we are now in a position to describe the main algorithmic steps undertaken by the GAs and the TEs while performing the Lookup List algorithm, thus in Algorithm 3 and 4, respectively. Recall that, since the *Resource Manager* is an always active component whose unique responsibility is to provide real-time information on node computational and communication availabilities, for the sake of simplicity we will not introduce it as a separate entity within the TE, but instead we will simply refer to it any time internal resource availability information is fetched within a TE node.

Unlike the Auction algorithm, in the Lookup List protocol, the GA opens Job_i execution by launching an initial *discovery phase* through which it discovers its neighbourhood and it fills a look-up list containing resource availabilities of in-range $\text{TE}_{y(a_{CPU},a_{Band})}$. In particular, the GA's *Main Thread* performs the following actions: (i) it broadcasts a request advertising the need for remote task offload; (ii) it waits Δt_1 time interval to receive availabilities from in-reach TE_y; (iii) it fills a look-up list containing resource availabilities of in-range $\text{TE}_{y(a_{CPU},a_{Band})}$. These actions iterate until at least N availabilities from TE_y are received. In fact, since the neighbourhood discovery phase is preliminary to Job_i execution and it is performed only at the beginning of the protocol, it is crucial to gather enough information about as many TE_y as possible in order to rely upon a bigger node selection while distributing tasks. In fact, while TE_y resource availabilities keep been constantly updated during the execution of the protocol, no other TE_y node is added to the list once the distribution takes place.

Upon receiving the offload request, in-range TE_y check with the internal *Resource Manager* component to determine their resource availabilities $\text{TE}_{y(a_{CPU},a_{Band})}$ and, after having collected them, they send such information to the GA. Once again, $\text{TE}_{y(a_{CPU})}$ identifies the number of available $\textit{TaskManager}_{available}$ in charge of handling remote task computation while $\text{TE}_{y(a_{Band})}$ identifies the local level of bandwidth availability and thus describes the level of network congestion of an area.

For each $Task_{i,j}$ that must be remotely computed, the GA needs to undertake a set of actions. Like the Auction algorithm, the GA needs to wait until there is at least one of its $\textit{TaskManager}_{available}$ available to handle a remote offload. Once this happens, the GA's *Main Thread* performs a computation to select the TE_{Best} from the previously filled look-up list.

Upon selecting TE_{Best}, the GA's *Main Thread* keeps sending a message to the TE_{Best} notifying its selection and waiting a time interval Δt_2 until it receives the ACK from the TE_{Best} confirming selection notification. Once it has received such a message, the TE_{Best} sends an ACK to the GA confirming selection notification together with an update of its internal resource availabilities $\text{TE}_{Best(a_{CPU},a_{Band})}$. The GA then updates the resource availability information contained within the look-up list and iterates the computation of the TE_{Best} selection. Upon updating the look-up list and further computing the best node selection, if the TE_{Best} keeps being the same node then $Task_{i,j}$ offload is performed on that very same node, otherwise the procedure iterates until a new TE_{Best} is selected. Once a positive ACK is re-

ceived, the GA offloads the preliminary data necessary to allow $Task_{i,j}$ computation on the TE_{Best} side. The GA then waits a time Δt_3 and offloads data until it receives the ACK from the TE_{Best} confirming the data offload reception. Upon receiving the ACK, the GA launches an independent *Task Manager* instance responsible to handle the remote task execution. Moreover, it updates its resource availabilities by increasing the number of $\textit{TaskManager}_{active}$ and decreasing those available $\textit{TaskManager}_{available}$.

Like the Auction algorithm, also within the negotiation phase of the Lookup List protocol, the exchanged messages are handled synchronously. This implies that no further action is undertaken within the protocol unless an ACK (either positive or negative) is received. Thus, if no ACK is received within a defined time interval Δt, the procedure iterates and a new message is delivered.

However, whenever *Task Manager* instances are launched on both the GA and the TE_{Best}'s nodes instead, they are individually and fully responsible for carrying on $Task_{i,j}$ execution. Thus, until $Task_{i,j}$ has reached completion, the GA and the TE_{Best}'s *Task Managers* communicate with each other every time it becomes necessary to exchange data required to support $Task_{i,j}$ computation.

Upon achieving $Task_{i,j}$ completion, the TE_{Best} asynchronously delivers the results to the GA and it further updates its internal resource availabilities $\text{TE}_{Best(a_{CPU},a_{Band})}$. In particular, the TE_{Best} increases the number of available $\textit{TaskManager}_{available}$, while decreasing the number of active $\textit{TaskManager}_{active}$ ones. Together with the $Task_{i,j}$ result, the TE_{Best} sends also an update of its internal resource availabilities $\text{TE}_{Best(a_{CPU},a_{Band})}$. As soon as $Task_{i,j}$ results are received at the GA side, the GA similarly frees its internal resources, thus increasing the number of available $\textit{TaskManager}_{available}$, while decreasing the number of active $\textit{TaskManager}_{active}$ ones. Moreover, it updates TE_{Best} resource availability information contained within the look-up list.

Whenever results for all tasks $Task_{i,j}$ of which Job_i is comprised have been computed, the GA computes the global Job_i result by combining all single task $Task_{i,j}$ results.

4.3 Bandwidth-Aware Task Scheduling Heuristic

As described in Section 4.1, whenever node collaboration and distributed computation is required to achieve Job_i completion, among the general characterisations of both task features $Task_{i,j(f_1,\ldots,f_F)}$ and the TE availabilities $\text{TE}_{y(a1,\ldots,a_A)}$, it becomes necessary to account particularly for two dimensions: computation (i.e. $Task_{i,j(f_{CPU})}$ and $\text{TE}_{y(a_{CPU})}$) and communication (i.e. $Task_{i,j(f_{Band})}$ and $\text{TE}_{y(a_{Band})}$).

In Section 4.2 we devised two algorithms to manage load distribution, namely the Auction and the Lookup List algorithms. Although different from a behavioural perspective (i.e. the Auction algorithm is reactive and GA-initiated, while the Lookup List algorithm is proactive and TE-initiated), both approaches have in common the need to perform a decision making phase in order to choose in an almost real-time way the best TE_y node towards which $Task_{i,j}$ offload needs to be directed.

Thus, each time the GA discovers its neighbourhood of TE_y nodes together with their internal resource availabilities $\text{TE}_{y(a_{CPU},a_{Band})}$, a utility function is computed in order to take the best informed decision about where the offload should be handled within the environment. In particular, we propose and integrate within the load distribution algorithms a *Bandwidth-Aware Task Scheduling* (BATS) heuristic.

BATS heuristic is built by linearly combining the information about the available computational resources of a TE_y node and the local amount of network contention performed in the area where TE_y is located. In Equation 4.1, we have that for each TE_y node, belonging to GA's neighbourhood dynamically discovered each time DWAG is applied, a score value $Score_{TE_y}$ is obtained by computing a linearly weighted score function combining computational $TE_{y(a_{CPU})}$ and communication $TE_{y(a_{Band})}$ availabilities of TE_y. $weight_{CPU}$ and $weight_{Band}$ are the weights set to define the relative importance of computational and communication resources, respectively. The TE_{Best} with the highest computed score $Score_{TE_{Best}}$ is selected to handle the remote $Task_{i,j}$ offload. In the experimental Section 4.6, we will describe how $TE_{y(a_{CPU})}$, $TE_{y(a_{Band})}$, $weight_{CPU}$ and $weight_{Band}$ are computed.

In bandwidth-unaware systems, only TE_y computational resources (i.e. $TE_{y(a_{CPU})}$) affect the selection, and thus $weight_{CPU} \neq 0$ and $weight_{Band} = 0$. On the contrary, in bandwidth-aware systems also the network traffic information (i.e. $TE_{y(a_{Band})}$) is taken into account when choosing the best candidate node towards which offload computation, and thus $weight_{CPU} \neq 0$, and $weight_{Band} \neq 0$.

$$Score_{TE_y} = (weight_{CPU} \cdot TE_{y(a_{CPU})}) + (weight_{Band} \cdot TE_{y(a_{Band})}) \tag{4.1}$$

$$Score_{TE_{Best}} = \max_{i=1}^{Y} Score_{TE_y} \tag{4.2}$$

Existing load distribution approaches (e.g. [Lu and Lau, 1996]) in traditional computational grid systems do not account for traffic contention levels in the underlying network while performing distribution, since they rely upon high-bandwidth reliable network links. On the contrary, since such reliable wireless radio links cannot be equally assumed for WSNs, we are interested in measuring how much they impact system performance in exploiting DWAG collaborations. For this reason, we proposed the BATS mechanism to account for bandwidth requirements as selection factor within the decision making process.

4.4 CPU and Bandwidth Load Computation

Before entering into the details of the evaluation, we describe the way we measure computational (CPU) and communication (bandwidth) load within WSNs. As described in Chapter 2, we evaluate our approach by using TMote Sky [TELOSB, 2010, Sentilla, 2010] ultra-low-power platforms programming them by adopting the Contiki Operating System (Contiki OS) [Dunkels et al., 2004], an open-source, highly portable, networked OS for memory-constrained systems. TMote Sky devices have been chosen for the evaluation since they represent an example of general purpose micro-controllers small enough to be portable in case of emergencies, but with enough computational power to perform basic tasks. Moreover, the research community in both academia and industry has greatly recognised and favoured their adoption and study since they are extremely cheap, easy to deploy and thus pervasively embedded within everyday life systems. However, while being so appealing from a form-factor perspective, their computational resources are sometimes scarce, especially when application sophistication increases. Node collaborations thus become necessary and with them the need to define both the computational load and

the local network condition information of the nodes towards which the offload needs to be performed. In our experiments, node computational load and local network contention are measured as detailed in the following sections.

4.4.1 CPU Load Computation

TMote Sky devices are equipped with a MSP430F1611 micro-controller from the TI-MSP430 family of ultra-low-power 8MHz 16-bit micro-controllers. As described in Section 2.2.1, we adopted the Contiki OS *protothreads* to implement multi-threading at the application level, since they are extremely lightweight to run on resource-constrained devices without leading to large memory overhead.

Whenever the execution of tasks in parallel is required, it is thus managed as follows. Each task is managed by a computational protothread. The state of different protothreads is stored in dedicated structures in memory, and every protothread is cyclically executed in a pre-defined immutable order. Furthermore, each time a protothread is called, its execution is never interrupted (i.e. no preemption) until the completion of a particular atomic, in-execution action (e.g. computation of a value, sending of a message, etc.) is achieved. As opposite to traditional multi-threading, the granularity of this particular context switching is thus very coarse, and the absence of a preemption mechanism leads to a very fair scheduling of the protothreads, where no one is given priority with respect to the others.

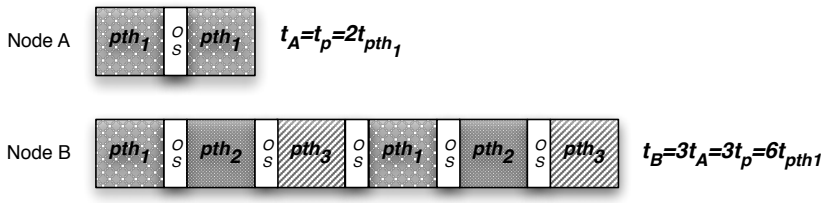


Figure 4.2: BATS computational load definition and computation. The number of protothreads executing at any one time on a node of the system represents a suitable approximation to measure node CPU load. Let us have nodes A and B, each of which required to execute a different number of protothreads with each of them homogeneous and purely computational. Node A executes one protothread (i.e. pth_1), node B executes three of them (i.e. pth_1, pth_2, pth_3). Node A will complete pth_1 in a time roughly equal to the number of functions (i.e. pth_1 has two functions) times the execution time of a single function (i.e t_{pth1}). Node B will need three times that interval. If we define the time necessary to complete one of the protothreads as t_p, then node A will take $t_p = 2t_{pth1}$ to finish its task, while node B will take $3 * t_p = 6t_{pth1}$.

Moreover, as opposite to a normal computer, the resources used by a user-defined protothread are much larger in proportion than those used by the underlying Contiki OS, which instead only adds a minimal overhead to the system, and thus the impact of running or not an additional task is much more considerable. For example, as illustrated in Figure 4.2, let us assume there are 2 nodes, A and B, each of which required to execute a different number of tasks, each associated to a protothread. In particular, each protothread is homogeneous, purely computational, and has to compute the result of a fixed list of mathematical functions. Node A has to execute one protothread (i.e. pth_1), while node B has to execute three of them (i.e. pth_1, pth_2, pth_3). In this case, node A will complete the execution of its task in a time which is roughly equal to the number of functions in the list (i.e. two functions for pth_1

in Figure 4.2) times the execution time of a single function (i.e t_{pth1}). Node B, instead, will need three times the same interval, since each of the protothreads will execute the first function, then pause while the others are doing the same, then execute the second function, and so on until all values have been computed. Therefore, if we define the time necessary to complete one of the protothreads as t_p, then node A will take $t_p = 2t_{pth1}$ to finish its task, while node B will take $3 * t_p = 6t_{pth1}$. Since node A is less loaded than node B, it is thus capable of starting another protothread while still being able to finish both before node B will complete its three ones.

Therefore, the number of tasks (each computed by a protothread) executing at any one time on a node has been chosen as the value to represent the CPU load of the node itself. In particular, $\text{TE}_{y(a_{CPU})}$ in Equation 4.1 has thus been set as the difference between the maximum number of protothreads, that we set to run tasks on a node, and those effectively running on it. Finally, recall that the responsibility of fetching low level information on the computational load of a node is handled by the *Resource Manager* introduced in Section 3.3.

4.4.2 Bandwidth Load Computation

As described in Section 2.2.2, TMote Sky devices communicate wirelessly with each other through the radio medium. They are equipped with a CC2420 module, an IEEE 802.15.4 compliant radio transceiver from Chipcon/Texas Instruments allowing a maximum bandwidth throughput of 250Kbps. In order to collect responsive, real-time, information on the level of traffic contention in the environment, we sampled over time the Clear Channel Assessment (CCA) value of the Chipcon CC2420 RF transceiver (i.e. corresponding to hardware pin 28 [Instruments, 2006] in Figure 4.3). We thus maintain over time a window of sampled values gathering information for the last n temporal slots and showing the times in which the probed radio channel is *clear* or *busy*. The information in the temporal window is used to estimate the network conditions of the local area in which the TE device is located: the more slots are busy, the greater is the probability that the node is located in a congested area.

We now discuss how to dimension the temporal window in order to capture a realistic snapshot and estimation of the TEs local network conditions when such conditions are time-dependent. Within the experimental evaluation, the dimension of the window has been tuned to $n = 100$ since this value has been able to simultaneously mediate the two following situations. The first case occurs when the value of the temporal window n is overestimated. In such a situation, whenever the GA needs to decide the best TE towards which to perform task distribution, the decision making process is at risk of being influenced by a considerable amount of out-of-date information thus preventing the system from reacting and adapting quickly enough to the local environmental changes. Similarly, when the value of the temporal window n is instead underestimated, the historical information incurs the risk of varying far too often, thus not effectively providing a meaningful picture of the TE local network conditions. Therefore, in order to represent a meaningful estimation of the TE's local network conditions when they vary with time, n needs to be tuned such that in the time interval between the end of a task offload and the beginning of a new decision making phase, the window needs to detect and provide an up-to-date vision of the local network conditions information.

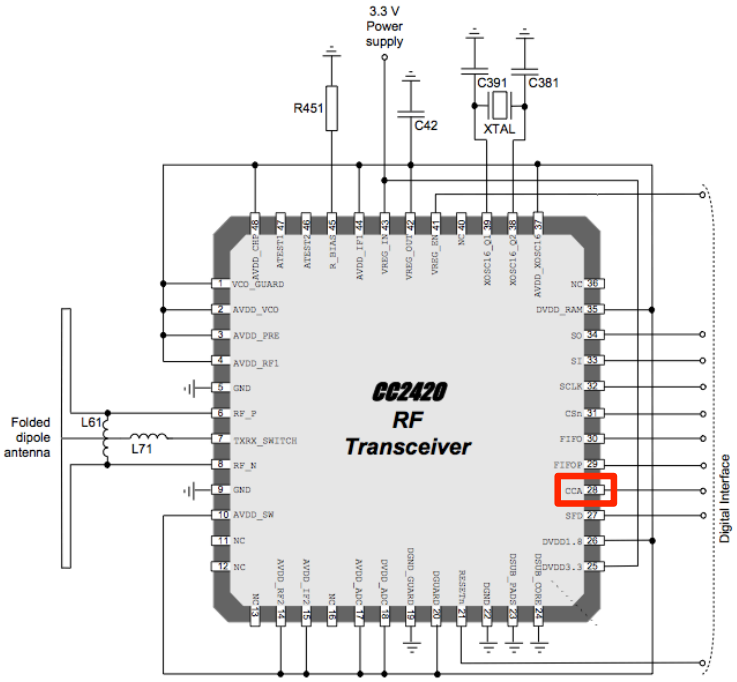


Figure 4.3: Clear Channel Assessment value (hardware pin 28) probed from Chipcon CC2420 RF transceiver.

For example, consider the situations illustrated in Figures 4.4 and 4.5. In both cases, the local level of network contention changes over time thus alternating periods of contention to others of no contention. In every time window, the radio channel is probed n times. In each instant of time t_i, the system computes the summarised network contention information S_i elaborating the n probes collected in the previous $\{t_{i-1}, t_i\}$ temporal window. The value S_i computed in t_i is thus maintained and used to make offload decisions in the following $\{t_i, t_{i+1}\}$ window.

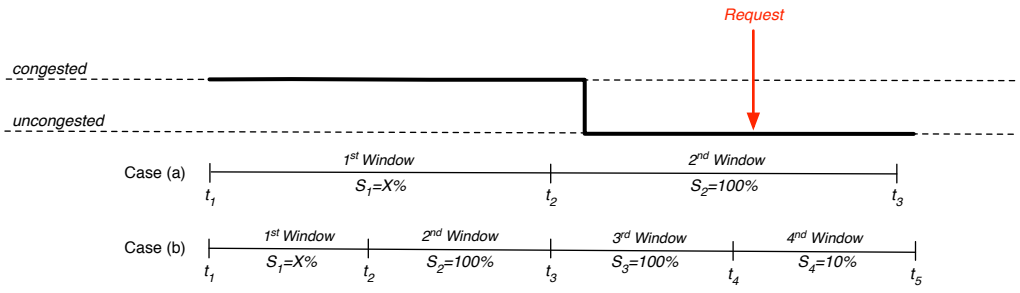


Figure 4.4: BATS bandwidth load definition and computation with overestimated temporal window. If the sampling window is overestimated, sudden network contention changes are not detected promptly and the out-of-date information maintained by S_i leads to inaccurate offload decisions (i.e. in case (a), $S_2 = 100\%$ computed in $\{t_1, t_2\}$ is erroneously maintained for the whole $\{t_2, t_3\}$, despite network trend inversions). A correctly estimated temporal window would promptly adapt to network changes, thus storing up-to-date trends and leading to correct offload decisions (i.e. in case (b), network inversion is detected quickly in t_4 and propagated in $\{t_4, t_5\}$).

As illustrated in Figure 4.4(a), if the sampling window is overestimated then sudden network contention changes are not detected promptly and thus the out-of-date information maintained by S_i leads

to inaccurate offload decisions. For example, the value $S_2 = 100\%$, computed in t_2 and synthesising the n probes performed in $\{t_1, t_2\}$, represents the information provided in $\{t_2, t_3\}$ to deal with any request (red arrow) coming from the application overlay aiming at fetching low-level information on network utilisation. However, because of the overestimated window, $S_2 = 100\%$ is kept for a relatively long time despite the network inversion. This thus leads the system to make inaccurate offload decisions. On the contrary, a correctly estimated temporal window, as illustrated in Figure 4.4(b), would instead promptly adapt to the network changes, thus storing up-to-date trends and leading to correct offload decisions. In fact, the network inversion is detected quickly in t_4 and propagated in $\{t_4, t_5\}$.

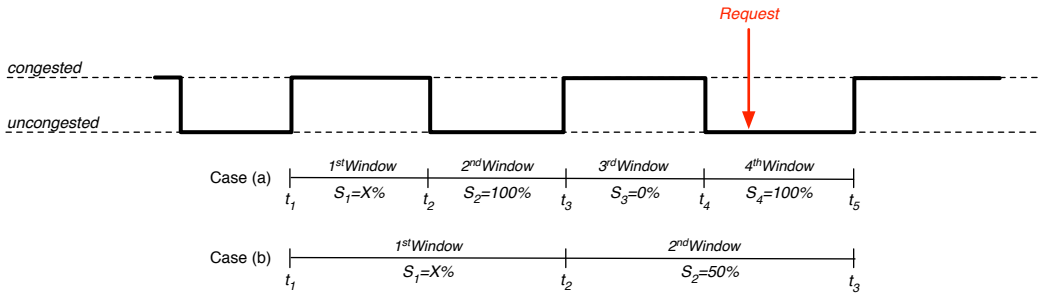


Figure 4.5: BATS bandwidth load definition and computation with underestimated temporal window. If the value n of the window is underestimated, the risk is that of feeding the application overlay (red arrow) at the request time with not representative values of the historical network traffic trend (i.e. in case (a), S_i rapidly switches between $S_i = 0\%$ and $S_i = 100\%$). A not underestimated window would be able to mediate such sudden changes (i.e. in case (b), a summarised value like $S_2 = 50\%$ is computed) identifying realistic traffic pattern, storing up-to-date trends and leading to correct offload decisions.

If the value n of the window is instead underestimated, as illustrated in Figure 4.5(a), then the risk is that of feeding the application overlay at the request time (red arrow) with values that are not truly representative of the historical network congestion trend. In fact, if the traffic rapidly oscillates, then S_i rapidly switches between $S_i = 0\%$ and $S_i = 100\%$, not truly representing the historical network trend. A correctly estimated window, as illustrated in Figure 4.5(b), would instead be able to summarise such sudden changes with a value like $S_2 = 50\%$ identifying a more realistic traffic pattern and thus guiding the system through correct offload decisions.

The bandwidth sampling process is iterated approximatively every $\frac{1}{64}$s, since it is periodically activated every time a Contiki OS timer expires. The summarised value S_i is updated after $n = 100$ probes have been collected and thus with a frequency of $\frac{1}{64}s * 100$. After running several experiments simulating network traffic, we noticed that the temporal window oscillated fast for $n <\sim 70$ probes and it was, instead, not promptly responsive for $n >\sim 130$. Thus, we tuned the number of probes for the temporal window to $n = 100$, since this value was able to provide a suitable estimation of local network traffic contention within our experimental settings. Recall that the sampling procedure is performed by an autonomous protothread, the *Resource Manager* component, running on each TE, and thus in a completely distributed way. In fact, as described in Section 3.3, each TE is formed by a single always active *Resource Manager* instance responsible for periodically fetching low-level information related to both the TE computational resources and local network conditions.

Since the value of $TE_{y(a_{Band})}$ in Equation 4.1 identifies the local level of available bandwidth, it is thus computed as the difference between the maximum number of samples collected within a window (i.e. $n = 100$ in our experiments) and the number of probes for which the sampled radio channel was detected to be *busy*.

4.5 Related Work

We are now interested in briefly discussing the reason behind the choice of the load sharing algorithms, namely Auction and Lookup List, selected and adapted within the work presented in the current dissertation.

4.5.1 Load Distribution Principles

In Section 4.2, we presented the two existing load sharing algorithms chosen, modified, and adapted to account for network utilisation and local traffic congestion, while performing DWAG. The Auction algorithm, described in Section 4.2.1, is a GA-initiated load sharing algorithm while the Looup List algorithm, described in Section 4.2.2, is instead inspired to the dynamic TE-initiated load sharing algorithm presented in [Chuang and Cheng, 2002]. We are now in a position to briefly discuss the reason behind the choice of the adopted load distribution algorithms.

There is a particularly extensive literature review on the computation distribution problem, thus the relocation of tasks from busy nodes to others that are lightly loaded, dating back twenty or more years [Eager et al., 1986, Ferguson et al., 1988, Lu and Lau, 1996].

In [Eager et al., 1986], the authors study, for example, the use of system state information in adaptive load sharing policies for locally distributed systems in order to determine an appropriate level of policy complexity. The results show that extremely simple adaptive load sharing policies, collecting considerably small amount of state information and using it in extremely simple ways, yield dramatic performance improvements compared to no load sharing cases. Moreover, results show how simple policies lead to performance gains similar to those achieved through complex policies, whose viability is however questionable. In [Ferguson et al., 1988], an approach based on concepts drawn from microeconomics is described, which uses algorithms that are competitive rather than cooperative. In particular, the authors present an economic approach in which (i) competition sets prices for the resources in the system, (ii) jobs compete for the resources by issuing bids, and (iii) the resource allocation decisions are made through auctions held by the processors. The benefits of the presented method include limited complexity and algorithms that are intrinsically decentralised and modular. The listed approaches represent only few examples of the most famous techniques introduced to cope with load sharing. These algorithms, even if particularly dated, attract for their embedded simplicity. In fact, although in the last years several algorithms, even more sophisticated, have been devised, work has continued more steadily, after an initial rush of enthusiasm, with an upturn in interest as part of computational grid activities. Within computational grids, most of the devised approaches tackle the problem of load distribution within more or less heterogeneous systems with ideally multiple task classes that are distributed in batch on several processors. For example, in [Lau et al., 2006] the authors propose a class of load distribution algorithms

that allow a batch of tasks to be transferred during each negotiation session. The core of the algorithms is a protocol that ensures a sender-receiver pair to negotiate and arrive at a suitable batch size. The protocol takes into consideration the processing speeds of the sender and receiver, as well as their relative workload, thus ensuring the maximal benefit for each negotiation session.

More recent approaches [Pontelli et al., 2010, Gmach et al., 2009, Qin et al., 2009, Shi and Kencl, 2006, Antonis et al., 2004] present complex infrastructures and schemes based on policies to balance the load of systems. However, the devised techniques are highly sophisticated, they are mainly developed for powerful clusters of nodes organised in a tree-hierarchical topology, they reply upon centralised authorities to decide on resource allocation, and they incur a non-negligible policy overhead.

A complete survey on meta-schedulers [Dong and Akl, 2006] for computational grids has been proposed. However, the approaches devised for such computational grids context are largely unsuitable for application to networks of resource-constrained devices for the following reasons. Firstly, they rely upon cumbersome infrastructure and frameworks running on powerful machines with often multi-core processors. Secondly, communication among nodes occurs through extremely high-speed network links, and not through wireless communication, thus communication does not play a crucial role or a bottleneck within the distribution process. Consequently, the focus is mainly related to optimise batch distribution and parallel computation exploiting processors multi-threading capabilities. Thirdly, scheduling techniques are mainly managed through a centralised authority responsible for resource allocation and load distribution. Fourthly, the devised infrastructures and algorithms are particularly sophisticated and demanding, thus not lightweight enough to run on devices with limited resources.

Recently, some approaches [Tipsuwan et al., 2009, Izakian et al., 2010] adopted auction techniques to distribute different kinds of load within the system. In particular, in [Tipsuwan et al., 2009] the authors propose a dynamic bandwidth allocation methodology, based on auction mechanisms, to control bandwidths given to open-loop networked control system to be at Nash equilibrium. However, the method still relies upon the presence of a centralised access point authority, acting as a broker between control and action agents, thus collecting information from the various resources and thus managing in a centralised way the actual allocation process. In addition, although simulation and experimental results showed the effectiveness of the proposed methodology, this was not practically implemented and evaluated in a decentralised way on standard wireless protocols. In [Izakian et al., 2010], the authors introduce a continuous double auction method for grid resource allocation in which resources are considered as provider agents and users as consumer agents. In their method, entities are allowed to participate in a grid independently and make decisions autonomously. However, although simulations illustrate that the proposed method is efficient in terms of successful execution rates, resource utilisation and fair profit allocation, it does not account for typical network problems arising within WSNs. Auction mechanisms [Lei et al., 2009] have been recently adopted within target tracking WSNs scenarios to create congestion control mechanisms.

Consequently, within the current dissertation we chose, adapted and implemented on deployed WSNs systems two simple load sharing algorithms. Simplicity is one of the major requirements because

of the limited resources available. The chosen algorithms do not claim to represent optimal allocation strategies but they, instead, aim at enhancing their fully distributed implementation and deployment while analysing the impact of network conditions in performing load distribution.

4.6 DWAG Evaluation with BATS Heuristic

In Section 4.4, we described the way we measured computational and communication load within WSNs. We are now in a position to detail the set of experiments undertaken within deployed systems to measure the impacts of traffic network congestion on the system performance while performing collaborative computation.

We evaluate our work according to both the work assumptions presented in Section 1.4 and the experimental criteria listed in Section 1.5. In particular, we apply the BATS heuristic to the DWAG paradigm. For the set of experiments presented in this chapter, Job_i is assumed to be split into a set of homogeneously characterised tasks $Task_{i,j}$ that must be distributed within the environment. In Chapter 5, we will instead focus our attention on task heterogeneity. In particular, we will evaluate the effects of offloading heterogeneous kind of tasks $Task_{i,j}$, of which every single Job_i is comprised, within the environment and we will therefore compare the BATS strategy against another heuristic dealing with task profile information.

In the first set of experiments presented in Section 4.6.3, we measure the impact of DWAG collaborations whenever a single GA node is exclusively entitled to perform distribution within the environment. In the second set of experiments presented in Section 4.6.4, we measure instead the impacts of such collaborations whenever multiple GA nodes simultaneously perform distribution within the environment and thus compete to achieve job computation. In Section 4.6.1, we define the general experimental setup while we clarify the peculiar characteristics of each experiment in Sections 4.6.3 and 4.6.4, respectively. Among the multitude of performed experiments, we decided to report in this dissertation the following findings since they are those corresponding to significant data variations.

4.6.1 Experimental Setup

We are now in a position to describe the general experimental settings for the experiments we undertake.

In this first set of experiments, the aim is that of comparing the different behaviours of the selected load distribution algorithms (i.e. Auction and Lookup List) in the presence of diversified levels of traffic and with either one or multiple GAs simultaneously interacting within the environment. The experiments are performed on the Heterogeneous Experimental Network (HEN) [HEN, 2010] TMote Sky sensor testbed deployed in the Department of Computer Science at University College London (UCL-CS). Details about UCL-CS HEN TMote Sky testbed are provided in Section 4.6.2. Recall that the choice of an office-like environment as experimental setting is in line with the emergency scenario presented in Section 3.1 (chosen among the set of motivating scenarios described in Section 1.1). In fact, the emergency events that we are interested in analysing are those occurring within indoor environments (e.g. formed by corridors, rooms and office-like furniture) in which multiple sources of wireless communication compete with each other generating medium contention. The UCL-CS HEN TMote Sky

testbed has been chosen because it displays such environmental characterising features. Moreover, in order to emulate the heterogeneous amount of communication expected for example during emergencies, we have introduced additional interferer nodes, as described in the following paragraphs.

Each node in the system is uniquely identified and different code is pre-loaded on nodes. In fact, while the GAs are loaded with Job_i, the TEs are instead pre-loaded with a set of functionalities aiming at supporting the GAs in their process of distributing the multiplicity of $Task_{i,j}$, of which each of their Job_i are comprised, in the environment to achieve Job_i completion. In fact, as described in Section 3.2, since DWAG has been designed in the first place to be independent from the underling loading code infrastructure and since a variety of related work (e.g. [Costa et al., 2007]) already exists in terms of middleware capable of on-the-fly loading and linking code, we confine ourselves to code pre-deployment within this dissertation.

Three main phases have been emulated: firstly, the communication offload of the data inputs from the GA to the TE; secondly, the inner communication exchanges needed to progress with task execution; thirdly, the upload of the computed result from the TE to the GA.

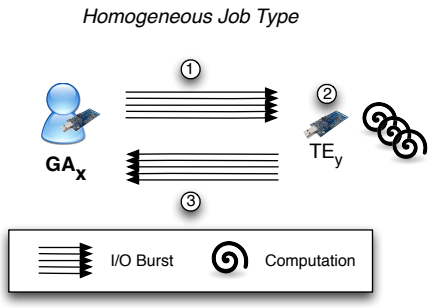


Figure 4.6: Homogeneous job type description.

In this experimental setting, we deal with homogeneous tasks. This means that (i) the number of messages offloaded from the GA to the TE and uploaded back from the TE to the GA is the same for every $Task_{i,j}$, and (ii) the amount of computation performed at the TE side is the same for every $Task_{i,j}$.

As illustrated in Figure 4.6, for each $Task_{i,j}$ to be computed, the GA offloads X Input/Output (I/O) bursts to the TE, each of Y bytes containing information to be processed, while the TE uploads X I/O bursts to the GA, each one of Y bytes containing the result from the computation. After running several experiments, we tuned the default task offload size value X to 50 and the dimension of the messages exchanged Y to 15 bytes. These values allowed, in fact, protothreads to interrupt each other in order to reproduce actual task concurrency (i.e. multiple tasks can run simultaneously on a TE exploiting concurrent execution, even in the simpler case of a GA and a TE) and, therefore, representing a situation common in parallel computing. Values $X <\sim 20$ and $Y <\sim 5$ bytes were leading, in fact, to a sequential flow of execution and not concurrency. Moreover, we tuned to 15 bytes the actual amount of exchanged information since this value was representative of the data exchanged through short bursts within collaborative localisation scenarios (i.e. node maps exchanges), as described in Chapter 6.

Each offload I/O burst contains the information Z to be processed. The value Z represents the number of times that a function is computed at the TE side. Since the tasks are homogeneous, the value Z adopted was kept constant within this experimental setup. We chose fibonacci as the function to be computed at the TE side, but recall that any other mathematical function could have been equally adopted within the experiments to engage the CPU in computational cycles. We tuned the value Z to 20: TE's CPU performs 20 times the fibonacci sequence of a number. This number was set to 15 (i) to allow Contiki OS protothreads to interrupt each other (within the Z cycles to be performed) exploiting concurrency, and (ii) to represent jobs in line with the logic of the real-world localisation case study presented in Chapter 6.

Message exchanges are managed by adopting the uIP TCP/IP stack implemented within the Contiki OS. This stack provides interoperability with existing systems and favours the integration of Contiki into existing IP network infrastructures. In particular, the GAs and the TEs handle both UDP and TCP forms of communication: the former is used to manage the negotiation phases characterising both the Auction and the Lookup List algorithms, whilst the latter is used to cope with the effective data offload/upload required to achieve remote computation. In fact, although the TCP protocol is heavier than UDP, we still need its reliability to be sure that every data packet is effectively communicated.

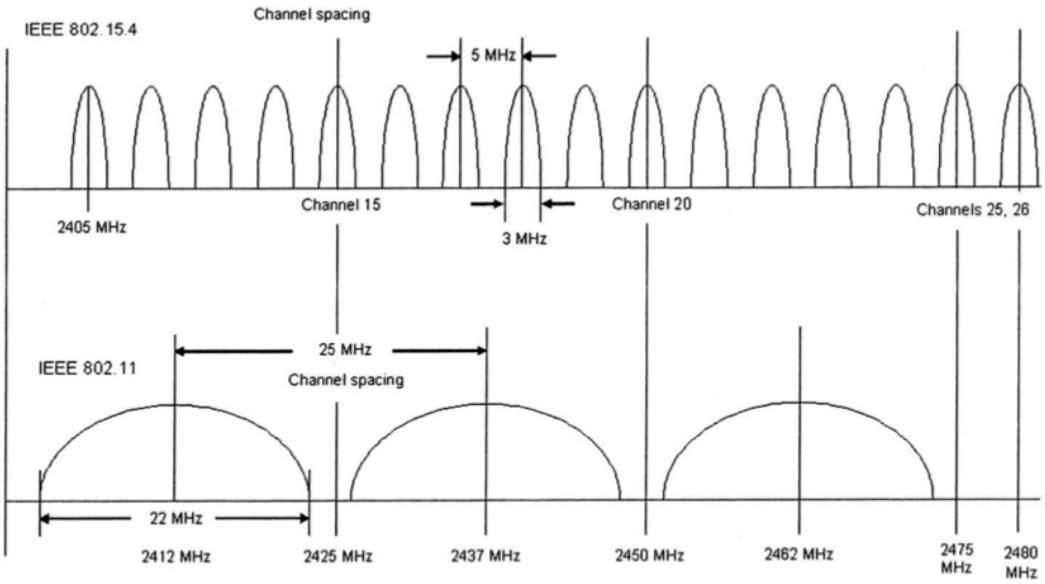


Figure 4.7: RF channel spectrum of IEEE 802.15.4/ZigBee against IEEE 802.11b/WiFi.

In the following experiments, we programmed TMote Sky devices by setting the Radio Frequency (RF) ZigBee channel to 26. This was done in order to minimise the radio interferences between IEEE 802.15.4/ZigBee and IEEE 802.11b/WiFi. In fact, as described in [Crossbow, 2007], the overlap of ZigBee and WiFi RF spectrum leads to slight radio interference. As illustrated in Figure 4.7, channel number 26 centred at 2480 MHz was chosen to handle WSN radio communications in order to minimise the frequency overlap between bands of near-operating WiFi systems, existing in the UCL-CS open-

space office where the HEN testbed was located.

As described in Section 3.3, both the GAs and the TEs concurrently handle a number of *Task Manager* instances representing the number of tasks executing in parallel on a node (i.e. the actual degree of parallelism). After a tuning phase, we set such value to 3 since up to this value the context switching among processes does not affect the computational resources of TMote Sky devices. As described in Section 4.4.1, parallel processes are implemented using the Contiki OS protothreads [Dunkels et al., 2006] and the communication among them is obtained by the possibility to simultaneously open and manage several TCP/IP connections through the Contiki OS protosockets.

We tested the DWAG paradigm with BATS mechanism with actual computation, actual profiling of the medium, and actual network traffic. In order to reproduce the situation described in Section 4.1, and thus to generate heterogeneous network traffic contention levels, we create a situation like that illustrated in Figure 4.8(a), where existing communication flows disseminating information from nodes A to B may indirectly affect the TE local network conditions, thus congesting the radio medium in the TE reception area. Whenever the TE is therefore selected to perform $Task_i$ computation, its performance might be influenced by the presence of such existing communication overhead. Within our experiments we reproduce such traffic contention by locating an additional sensor node, namely the interferer node Str in Figure 4.8(b), within the TE radio range while at the same time limiting Str radio transmission range in order for it to be heard only from a subset of nodes (i.e. part of TEs) and not from others (i.e. GAs). Note that Str reproduces the situation of having heterogeneous network contention but within the same MAC layer. However, since the experiments are performed on a deployed testbed within an office, interference coming from additional sources of network communication (e.g. IEEE 802.11b/WiFi) not acting within the same MAC adds itself to that generated by Str.

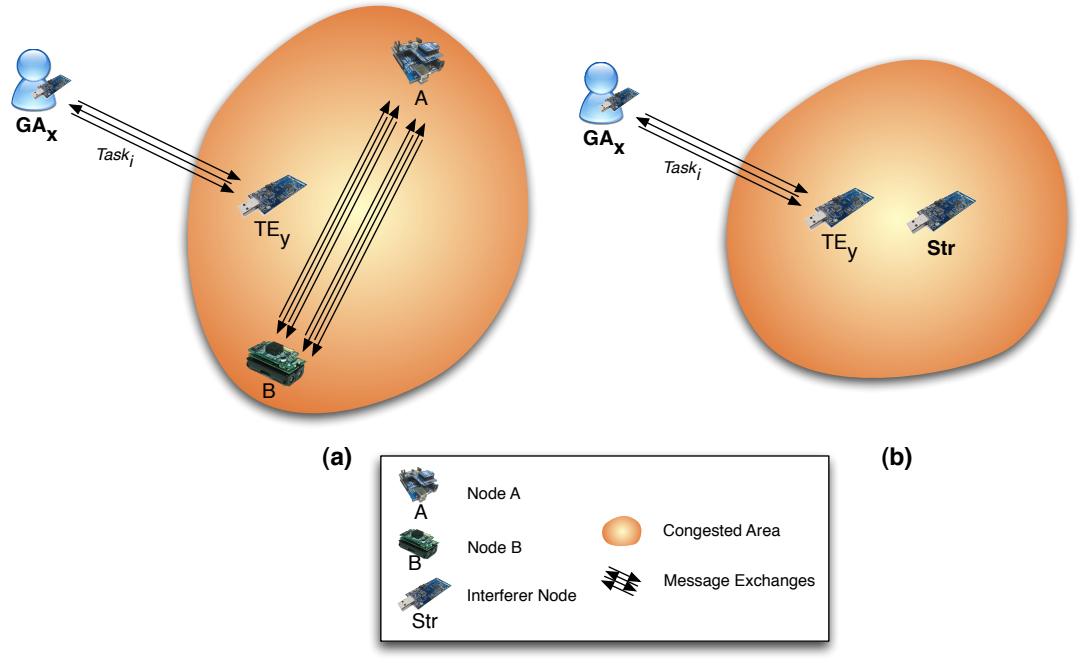


Figure 4.8: Introduction of an interferer node to handle heterogeneous network traffic contention.

To gauge the appropriate rate at which the Str node keeps disseminating messages, we performed an experiment as illustrated in Figure 4.8(b). Firstly, we positioned the Str node within the TE's radio range and we set Str radio transmission power to the second lowest possible level (i.e. -28 dBm), so that messages sent by Str could exclusively influence the TE, but not the GA. Then, we instructed the GA to send 100 messages towards the TE. The messages were sent in a best effort manner, without attempting any retransmission if any of them was lost. The Str node was configured so that bursts of 3 bytes were injected in the network after each time a trivial computation was executed. We thus managed to fine tune the Str node, so that approximately 75% of the messages sent by the GA were received by the TE. The experiment was repeated 20 times, changing the positions of the GA and the TE. Notice that in the real experiments, whenever a message is lost it needs to be retransmitted, thus delaying the whole offload process. In summary, we devised this simple experiment to tune the message transmission rate for the Str node introduced in each of the following experiments to create heterogeneous environmental traffic contention.

In order to warranty that the GAs were not affected by the additional communication injected by the interferer node Str, before launching each of the experiments and, thus, before activating the TEs, we instructed the GAs to gather sampled values over a temporal window showing the times in which the probed radio channel was *clear* or *busy* and to record such values on a log. The presence of a detected *busy* radio channel translated into the GAs being affected by Str injected network contention. In this situation, we progressively varied the Str radio transmission power and position until the GAs were not affected by Str influence.

In order to apply the BATS heuristic described in Equation 4.1, it is necessary to compute not only $\text{TE}_{y(a_{CPU})}$ and $\text{TE}_{y(a_{Band})}$ values, but also to set $weight_{CPU}$ and $weight_{Band}$. In Section 4.4, we described the process of computing both computation ($\text{TE}_{y(a_{CPU})}$) and communication ($\text{TE}_{y(a_{Band})}$) availabilities. $\text{TE}_{y(a_{CPU})}$ is the difference between the maximum number of protothreads (i.e. maximum number of parallel tasks) that we allow to run on a TE and those effectively running on it at the request time. $\text{TE}_{y(a_{Band})}$ instead is the difference between the maximum number of samples collected within a time window, $n = 100$ in our experiments, and the number of probes for which the sampled radio channel was detected to be *busy*.

$\text{TE}_{y(a_{CPU})}$ falls into range [0,3], because the maximum supported parallelism is 3, while $\text{TE}_{y(a_{Band})}$ falls into range [0,100], because of the number of probes ($n = 100$) collected by the TE. However, usually the bandwidth availability never drops below 50% (even in the case in which the Str node is congesting 25% of it, and the other nodes are using another 25%), so the range is better estimated as [50-100]. To normalise the range [50-100] within [0,50], a value of 50 was subtracted from $\text{TE}_{y(a_{Band})}$. Thus, to normalise both contributions $\text{TE}_{y(a_{CPU})}$ and $\text{TE}_{y(a_{Band})}$ within the same range [0,50], the weights $weight_{CPU}$ and $weight_{Band}$ were tuned to 16 (i.e. with the maximum value being $3 \cdot 16 = 48$) and 1 (i.e. with the maximum value being $1 \cdot 50 = 50$), respectively. The values are normalised according to the experimental setting and thus independent from the level of congestion.

Finally, in the following experiments we adopt latency as metric to measure the system performance

since it captures the effects that tasks from one node have on the execution patterns of others. We explicitly do not consider energy efficiency as a metric because, in the kind of scenarios we investigate, battery lifetime is much less of an issue than is timely information. In fact, information from sensors will be at its most useful within the first few tens of minutes of a happening, but in that timeframe it could allow prompt and earlier intervention. In the experiments, we compute the *job duration* intended as the overall time necessary for the totality of tasks, into which Job_i has been split, to reach completion. The algorithms were run 30 times, more if necessary to obtain statistically valid results. Each point in the graphs represents the average of all the runs and the error bars show the standard deviation over the runs. In each experiment, we vary the values of one input parameter and assign the default values to the others, if not mentioned otherwise.

In the experiments, we compare the performance of the load distribution Auction and Lookup List algorithms analysing the impact of network conditions during distribution by adopting the BATS scheme. In particular, we collect and analyse data coming from situations as follows:

- ***No Interferer (***$weight_{CPU} \neq 0$ ***and*** $weight_{Band} = 0$ ***in BATS)***:
  In this situation, we assume the presence of a homogeneous amount of network traffic in the network. No interferer node Str is present within the experiments and the only traffic is that occurring because of job distribution. In addition, whenever distribution must be undertaken, the decision is based on the TE computational capabilities. In short, this situation translates into no additional contention with $weight_{CPU} \neq 0$ and $weight_{Band} = 0$ in BATS.

- ***Interferer with No Bandwidth Control (***$weight_{CPU} \neq 0$ ***and*** $weight_{Band} = 0$ ***in BATS)***:
  In this situation, we introduce the interferer node Str into the system in order to generate additional network traffic. However, whenever distribution is required, the offload decision is based only on the TE computational capabilities. In short, this situation translates into distributing under heavy communication load, while ignoring the TE local network traffic information, and thus exclusively accounting for the TE computational availability, hence setting $weight_{CPU} \neq 0$ and $weight_{Band} = 0$ in BATS.

- ***Interferer with Bandwidth Control (***$weight_{CPU} \neq 0$ ***and*** $weight_{Band} \neq 0$ ***in BATS)***:
  In this situation, the system is again affected by the presence of the interferer node Str responsible for injecting traffic contention. However, in this case the offload decision accounts for the evaluation of both factors: TE local computational and network availabilities. In short, this situation translates into distributing under heavy communication load, while combining information regarding both the TE local network conditions and internal computational availabilities, hence setting $weight_{CPU} \neq 0$ and $weight_{Band} \neq 0$ in BATS.

We are now in a position to detail and analyse the results for each of the experiments performed.

4.6.2 UCL-CS HEN TMote Sky Sensor Testbed

As described in Section 4.6.1, the experiments reported in this dissertation have been mainly performed on the Heterogeneous Experimental Network (HEN) [HEN, 2010, Tuy and Muyal, 2009] TMote Sky

sensor testbed deployed at the Department of Computer Science at University College London (UCL-CS).

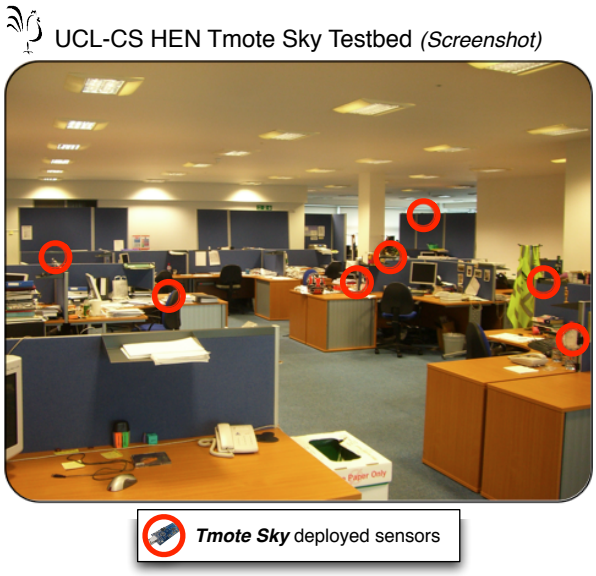


Figure 4.9: View of the UCL-CS Heterogeneous Experimental Network (HEN) TMote Sky sensor testbed.

The original HEN testbed comprised a set of PCs connected in clusters with the capability of being re-configured, running dual-stack, and emulating routers, servers, gateways, and workstations. It now consists of 80-100 nodes, each with multiple communication interfaces, thus it can be considered as a general-purpose network experiment testbed. Several interfaces can be configured as, or attached to, routers, mobile routers, 802.11 APs, other embedded systems. Open-source gateways supporting multiple network technologies such as Ethernet, WiFi and IEEE 802.15.4 have been enhanced with protocol stacks suitable for wireless sensor Internet working and basic multimedia capabilities. There is also glue software for management of resources. All the nodes can be configured as dual-stack. Some of the internal wireless sensors run only IPv6, whereas external routers and servers attached run dual-stack. HEN has been recently adopted within U-2010 [U-2010, 2010] and RUNES [Costa et al., 2007] projects.

HEN is used for many types of experiments. One of these is with an attached WSN which can be configured with IPv4, IPv6 or dual-stack. The IPv6 WSN testbed subset consists of 40 fixed TMote Sky platforms. As shown in Figure 4.9, the UCL-CS HEN TMote Sky testbed spans a large laboratory area with realistic radio conditions. Moreover, it has several strengths: firstly, platforms are randomly deployed in order to eliminate any source of behavioural predictability; secondly, sensors can be remotely accessible and programmable through fast kernel flashing able to load binaries on single devices; and, lastly, it represents a large-scale environmental testing source. The devices run sensor Operating Systems such as TinyOS [System, 2010b] and Contiki OS [Dunkels et al., 2004]. The whole system is thus a fully distributed network used for all sorts of network research in the UCL-CS department, usually not IPv6 enabled. The facilities include those for development, deployment, administration, management, experimentation and testing of WSN networks. Work in the area of IPv6-based sensor testbeds

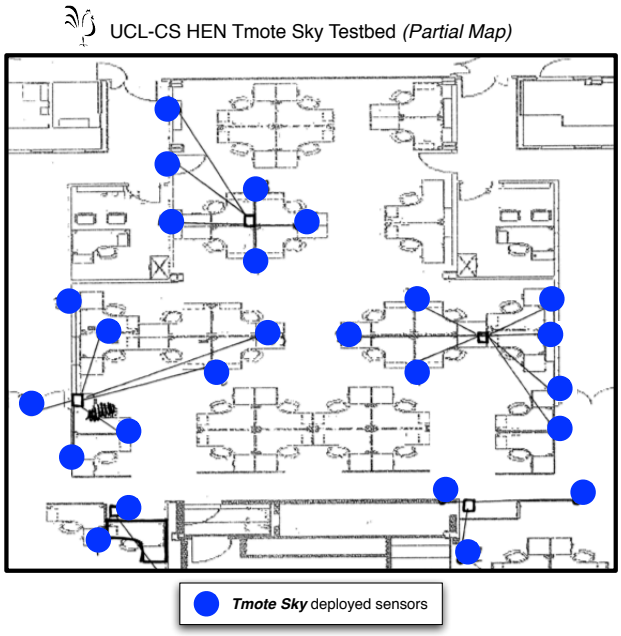


Figure 4.10: Partial map of the UCL-CS HEN device deployment.

centres on efficient monitoring of tunnels and buildings, wide area IP-based network surveillance, and response to emergency situations triggered by sensors deployed in such constructions. One can cross-build mote applications or other middleware on UCL-CS HEN, or can upload mote binaries developed by others. Figure 4.9 displays a snapshot of the the actual open-space area where devices are located, while Figure 4.10 shows a partial map of the actual device distribution within the open-space floor.

The goal of UCL-CS HEN as a general-purpose network is to bridge the gap between simulation and system deployments. In fact, too often network research has been confined to simulation, sometimes resulting in an over-simplification of the problem or doubt about the validity of results. The alternative was either very small-scale testing using desktop machines, or system deployment. HEN has thus been a valuable attempt to bridge the two drawbacks since (i) experiments can be large enough to be interesting, but (ii) the network environment is still under the researcher control, leading to a better understanding of real-world behaviours.

4.6.3 Experimental Results with Single Grid Activator

We are now in a position to describe the setting for each one of the experiments performed and to reason on the results achieved.

The aim of this first set of experiments is the investigation of the system behaviour in the presence of DWAG, and thus in dealing with distribution and collaborative computation, whenever a single GA node is required to exclusively perform within the environment. The experimental topology is as shown in Figure 4.11. In particular, for this experiment we selected 7 devices from the UCL-CS HEN TMote Sky testbed and instrumented them to run the Contiki OS binaries. In this context, only a single GA, namely GA_1 in Figure 4.11, is responsible to offload tasks to 5 possible TE_y (i.e. $\{TE_1, \ldots, TE_5\}$).

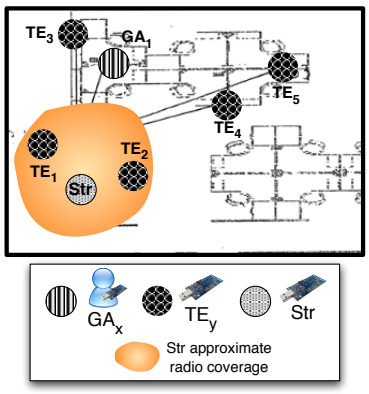


Figure 4.11: Map of UCL-CS HEN device deployment for experiments with single GA.

Whenever the interferer node Str is active, thus generating traffic perturbations and consequent heterogeneous network contention, the environment finds itself subdivided into two areas: the first contains nodes $\{TE_1, TE_2\}$ since they are located in a more network contended area; while the second contains nodes $\{TE_3, TE_4, TE_5\}$ that, not falling within Str's radio transmission range, are thus slightly affected by extra traffic. In the experiments, Str radio transmission power has been set to the second lowest level (i.e. -28 dBm in [Instruments, 2006]) subdividing the environment in two virtual areas of influence containing nodes $\{TE_1, TE_2\}$ and $\{TE_3, TE_4, TE_5\}$, respectively. As described in Section 4.6.1, in order to inject traffic contention, Str was configured so that bursts of 3 bytes were injected at regular intervals. Finally, recall that both the GAs and the TEs radio transmission power was kept to the original Contiki OS default value (i.e. 0 dBm in [Instruments, 2006]) to allow nodes to be in reach of each other.

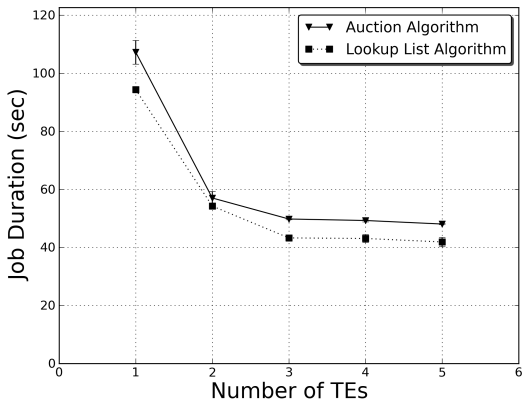


Figure 4.12: Effects of varying TE number with single GA experimentation.

Effect of varying the number of the TEs: The aim of the first experiment is to show the nature of the speed-up obtained if the number of the TEs available for computation increases by keeping constant the number of GAs distributing computation.

In this experiment, Job_i is comprised of 10 $Task_{i,j}$ required to be distributed from GA_1 to 5 possible TE, thus $\{TE_1, \ldots, TE_5\}$, adopting the Auction and the Lookup List algorithms without bandwidth

control mechanism (i.e. $weight_{CPU} \neq 0$ and $weight_{Band} = 0$ in BATS). In this situation, the Str node is inactive. Therefore, the TEs find themselves located in an environment homogeneous from a traffic perspective with the only possible being generated from distribution. As described in Section 4.6.1, we adopt homogeneous tasks. This means that (i) the number of messages offloaded from the GA to the TE and uploaded back from the TE to the GA is the same for every $Task_{i,j}$ (i.e. 100 I/O bursts each of 15 bytes in this experiment) and (ii) the amount of computation performed at the TE side is the same for every $Task_{i,j}$ (i.e. for each computational cycle, TE's CPU performs the fibonacci sequence of a number as mentioned above).

Experimental results, shown in Figure 4.12, display the nature of the speed-up obtained when the number of the TEs available for computation progressively increases from 1 to 5. As one can notice, the biggest performance improvements are detected in the situation in which the TEs become operational up to a number of 3. After this case in fact, the system reaches stability and additional TEs do not bring further benefits. This happens because, as described in Section 4.6.1, the degree of parallelism supported by both the GA and the TE nodes is 3. Let us now explain such behaviour as follows. Whenever the system is composed of one GA and one TE only, the TE is required to manage parallel computation and thus its resources are shared among all $Task_{i,j}$ simultaneously running on it thus slowing down the execution of each single task and causing longer completion times. As soon as more TEs are added to the system, extra resources become available: several $Task_{i,j}$ are thus distributed on a multiplicity of TEs, and each TE is not required to use its full capacity by sharing its own resources among several process executions. This results a speed-up of the whole job execution process.

It can be observed in Figure 4.12 that both load distribution algorithms show a similar execution trend with the proactive, TE-initiated Lookup List algorithm slightly outperforming Auction. This happens because whenever only a single GA is acting within the system, it is the only node responsible for actively modifying the status of TEs' execution while at the same time keeping stored within itself up-to-date data structures describing TEs' availabilities in real-time. In fact, since such availabilities are not affected by the concurrent activity of several GAs simultaneously load distributing applications within the environment, the up-to-date stored information allow the GA to make, in each instant of time, the best possible decision about where to perform the offload with very little overhead since the data structures are filled once and for all at the beginning of the algorithm and they are only updated at run-time. This thus determines the slight performance improvements achieved by the Lookup List algorithm as compared to Auction.

Effect of varying the task offload size: The aim of the second experiment is to show the impact of varying the number of streamed messages during both task offload and result upload phases on the algorithms performance.

In this situation, the interferer node Str is active, and it is thus responsible for injecting extra traffic within the environment. Moreover, network traffic is also varied through the different amount of network packets disseminated during $Task_{i,j}$ distribution. Again, Job_i is comprised of 10 $Task_{i,j}$ that must be distributed from GA_1 to 5 possible TEs, $\{TE_1, \ldots, TE_5\}$, adopting either the Auction or the Lookup

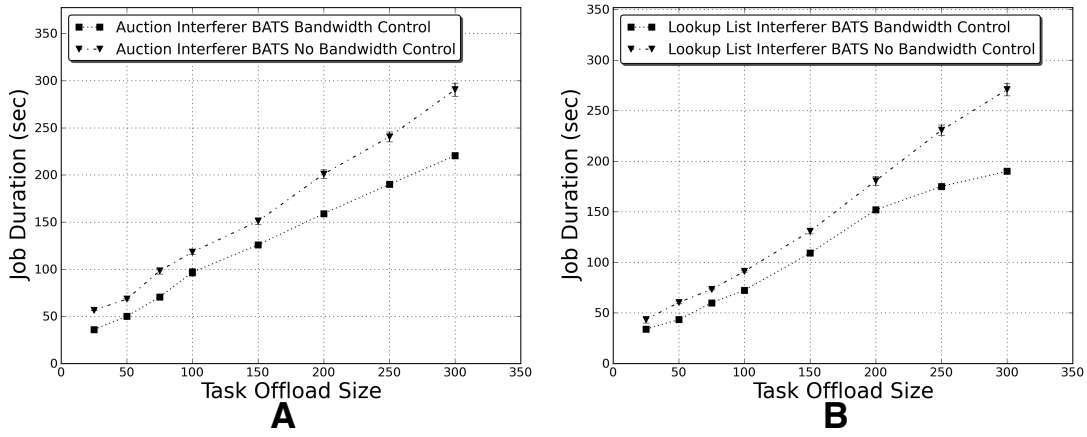


Figure 4.13: Effects of varying task offload size with single GA experimentation.

List algorithm. In the experiment, we vary from 25 to 300 the number of messages streamed both during the task offload and the result upload phases. In particular, for each $Task_{i,j}$ of which a job is comprised, we keep constant the number of messages to be streamed. Thus, from 25 to 300 message streams are delivered during both task offload and upload phases. We measure the performance of the Auction and the Lookup List algorithms in both situations in which, within the BATS heuristic, local network conditions are accounted or disregarded.

Experimental results for the Auction and the Lookup List algorithms are reported in Figure 4.13A and B, respectively. As can be seen, the increase in the number of packets to be exchanged for each task to be distributed leads to increased contention levels within the environment. In this situation, it becomes interesting to analyse the impact of making informed decisions accounting for both computational availability and network conditions. As shown in Figure 4.13A, performance improvements of $\sim$33% and $\sim$25% are detected for the Auction algorithm in the two extreme analysed situations in which 25 and 300 messages per task are load distributed, respectively. Similarly, performance improvements of $\sim$23% and $\sim$30% are detected for the same two cases by the Lookup List algorithm, as shown in Figure 4.13B.

Analysing Figures 4.13A and B together, the summary of displayed results provides a zoomed view over the single Auction and Lookup List algorithm behaviours. In particular, experimental results show that, although with similar behaviour and patterns, the Lookup List algorithm slightly outperforms Auction. The reason is as before. Whenever a single GA acts in the system, it is the only one responsible for actively affecting the environment modifying its level of traffic and the load of the nodes. However, in making decisions, it is always able to deal with mostly up-to-date kind of information since it is the only node injecting modifications within it. Thus, GA's structures record up-to-date information about the TEs. Such structures are created by the GA just once, at the beginning of the algorithm, and they are only updated during the algorithm execution. Moreover, a single GA load distributes intensive tasks and thus no multiple GAs concurrently access and modify TEs resources. For these reasons and since the Lookup List algorithm incurs very little overhead, it slightly outperforms the Auction algorithm.

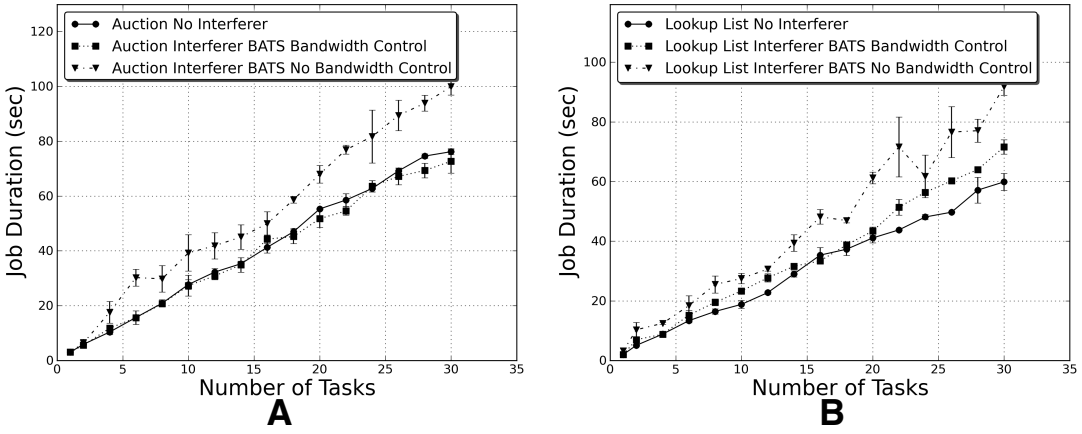


Figure 4.14: Effects of varying task number with single GA experimentation.

Effect of varying the number of tasks: The aim of the third experiment is to show the impact of varying the number of tasks of which a job is comprised on the performance of the algorithms.

For this experiment, we vary the number of tasks from 1 to 30. Distributed tasks are homogeneous and, as described in Section 4.6.1, they alternate streams of message offload with remote computation and result upload. In particular, 50 message streams (default value) are delivered during both offload and upload phases. In this experiment, we measure the performance of both the Auction and the Lookup List algorithms in terms of *job duration* and thus as the time required by GA_1 to achieve Job_i completion allowing for remote execution of up to 30 $Task_{i,j}$ to be load distributed on 5 possible nodes.

As described in Section 4.6.1, we compare for each load distribution algorithm the three situations of (i) *No Interferer*, (ii) *Interferer with No Bandwidth Control*, and (iii) *Interferer with Bandwidth Control*.

Experimental results in Figure 4.14A show that for the Auction algorithm, performance in the presence of contention is poor unless one considers bandwidth during task allocation, in which case system performance approximates that of the no interferer, uncongested, case. This is as expected since in the case in which GA_1 offloads a task towards a relatively uncongested zone, we expect overall performance close to the situation in which there is no congestion. In the case in which a task is sent towards a more congested zone, the same contention for the medium as that we have in the offload process is experienced. Moreover, during the execution of the task, contention for the medium will slow down the task computation itself relative to an uncongested situation, and the additional congestion generated will slow down other nodes executing in the same area. The error bars show a larger variation in the case of congestion without bandwidth control. On the other hand, the latency variations are minimal in the case of congestion with bandwidth control. This is as expected. In the last case, the predictability of task assignment is high since it will mostly go towards those TEs not belonging to congested environments.

As illustrated in Figure 4.14B, similar results are obtained for the Lookup List algorithm. Even in this case, as the error bars show, the performance variations are minimal for the case with interferer and

bandwidth control because the algorithm is capable of redirecting tasks to those TEs belonging to uncongested environments, thus avoiding the delays occurring by sending tasks to congested areas. However, the overall performance of the Lookup List algorithm is slightly better than Auction. We believe this is due to the inherently adaptive nature of the TE-driven approach to the availability of advertisements. After a short while, only uncontended nodes will advertise their availability for execution, eliminating the further source of contention present in the Auction algorithm.

4.6.4 Experimental Results with Multiple Grid Activator

We are now in a position to test our algorithms with a multiplicity of GAs. In particular, for the performed experiments we used a subset of 25 TMote Sky devices belonging to the UCL-CS HEN setting 3 GAs, 1 interferer node Str and 21 TEs, respectively.

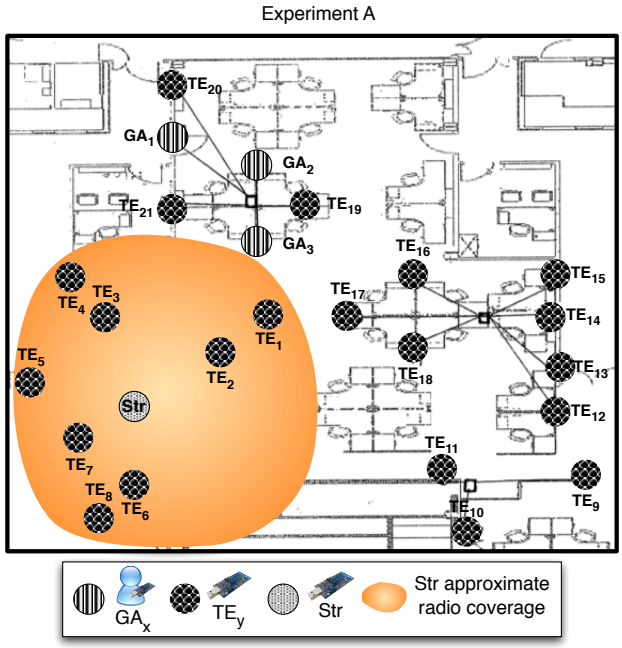


Figure 4.15: Map of UCL-CS HEN device deployment for Experiment A with multiple GA.

We set the radio transmission level of the interferer sensor Str to be -28 dBm in order to congest one part of the involved area, and thus producing a situation of heterogeneous traffic contention. As described above, in order to inject further traffic contention, Str node has been configured so that bursts of 3 bytes each were injected in the environment at regular intervals. Every Job_i to be performed is composed of 32 $Task_{i,j}$. We performed two set of experiments, namely *Experiment A* and *Experiment B*, varying the network topology. In particular, Figure 4.15 and Figure 4.17 show maps of selected nodes, together with their associated role, for both *Experiment A* and *Experiment B*, respectively. Experimental results collected for both *Experiment A* (i.e. Figures 4.16A and B) and *Experiment B* (i.e. Figures 4.18A and B) are obtained by averaging the job latencies gathered by each of the 3 GAs simultaneously distributing jobs within the environment. Moreover, as described in Section 4.6.1 the latencies for each GA are computed by performing the mean of approximatively 30 different experimental runs. We again compare for each load distribution algorithm the three situations described in Section 4.6.1.

In particular, the interest was in analysing and understanding the complex dynamics behind the two load sharing algorithms whenever a multiplicity of GAs is simultaneously distributing load within the system, thus actively modifying the environmental network traffic. In this situation, since several GAs are concurrently offloading tasks at the same time, not only does the level of network contention differ from the case where a single GA distributes load, but it also becomes crucial to analyse the algorithms performance in dealing with more up-to-date and out-of-date network conditions. For this reason, we exclusively present results and analyse the effect of varying the number of tasks, since varying either TEs number or task offload size leads to experimental results similar to those already showed in Section 4.6.3 for single GA experimentation.

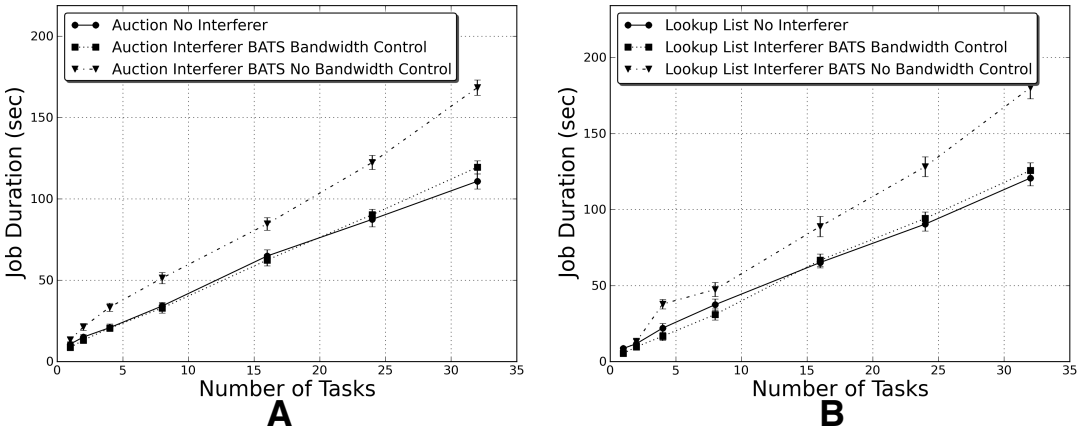


Figure 4.16: Effects of varying task number in Experiment A and multiple GA.

Effect of varying the number of tasks: The aim of the experiment is to show the impact of varying the number of tasks of which a job is comprised on the performance of the algorithms.

In the first experiment, we evaluate the impact of network contention on task distribution. Therefore, we adopt the two load sharing algorithms, Auction and Lookup List, together with the BATS heuristic but handling the distribution by making a decision exclusively accounting for TEs computational capabilities. We compare the two situations where: (i) no interferer node Str is present; (ii) extra traffic is injected within the environment by means of an interferer node Str.

Experimental results show that the additional network contention leads to an increase in the whole execution latency, with respect to the case without congestion or an homogeneous amount of communication, of $\sim$55% and $\sim$40% (Figures 4.16A and 4.18A) in the case of the Auction algorithm and $\sim$50% and $\sim$25% (Figures 4.16B and 4.18B) in the case of the Lookup List algorithm. This occurs when considering the simplest form of task offload to physically adjacent neighbours, and it represents an upper-bound on performance of WSNs, where similar problems of contention will be faced for each hop in a possible multi-hop route. Therefore, from this experiment we observe that the distributed computation may encounter severe performance degradations if bandwidth requirements are disregarded during task distribution.

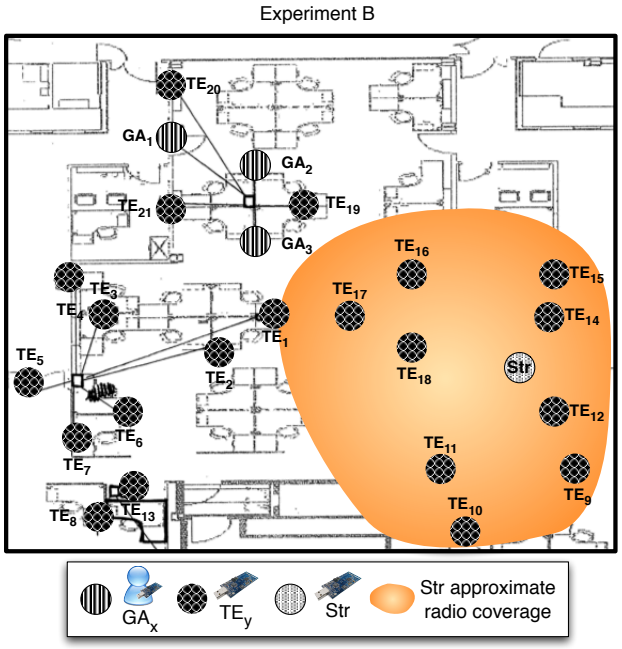


Figure 4.17: Map of UCL-CS HEN device deployment for Experiment B with multiple GA.

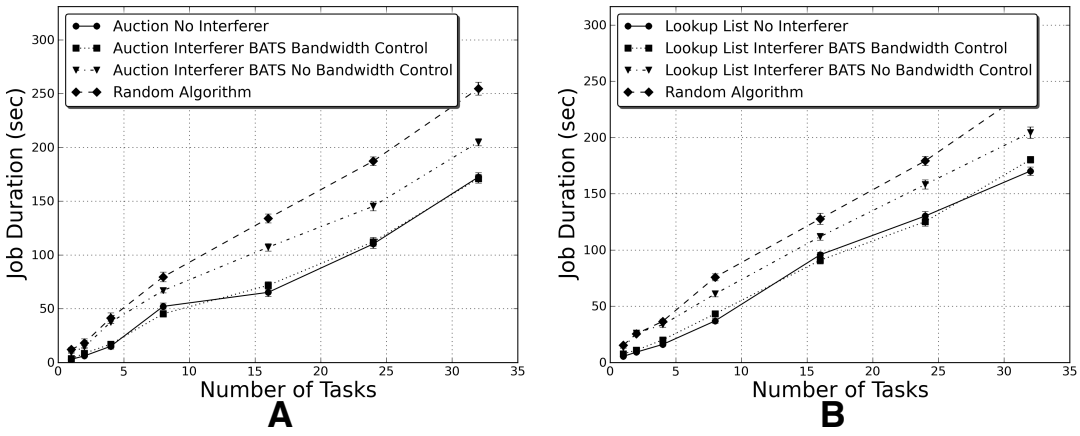


Figure 4.18: Effects of varying task number in Experiment B and multiple GA.

We then explore whether taking into account bandwidth availability has a significant impact on the *job duration*. Both *Experiment A* and *B* are performed with the same configuration as above but feeding the BATS heuristic with network information as well as computational load. Experimental results demonstrate that by taking TEs' local network conditions into consideration, thus combining both the TE computational capabilities with network contention information, system performance approximates that of the uncongested scenario. In fact, experimental results with multiple GAs confirm trends presented within single GA results: performance in the presence of contention is poor unless one considers network information during task allocation, in which case system performance approximates that of the uncongested case. Thus, a considerable improvement in latency is gained by accounting for local network conditions, while distributing computation.

However, a major difference emerges in comparing the behaviour of the Auction and Lookup List algorithms in the current experiments, if compared to those described in Section 4.6.3. In fact, if the TE-initiated algorithm (Lookup List) slightly outperforms the GA-initiated one (Auction) whenever a single GA is exclusively entitled to distribute load in the system, the opposite trend is displayed when multiple GAs concurrently distribute tasks within a shared environment, as illustrated in Figures 4.19A and B. This happens because when one GA exclusively offloads tasks into a pool of TEs, it is the only entity able to actively change the TEs status and so its lookup list can always effectively match in almost real-time TEs load. On the other hand, when several GAs independently offload tasks into the environment, the load of each TE can be modified by a GA without the other GAs being actively notified of such load modification. For this reason, such behaviour leads to progressively increases in gap between the information stored into the lookup lists at the GA's side and the real-time load of the TE nodes.

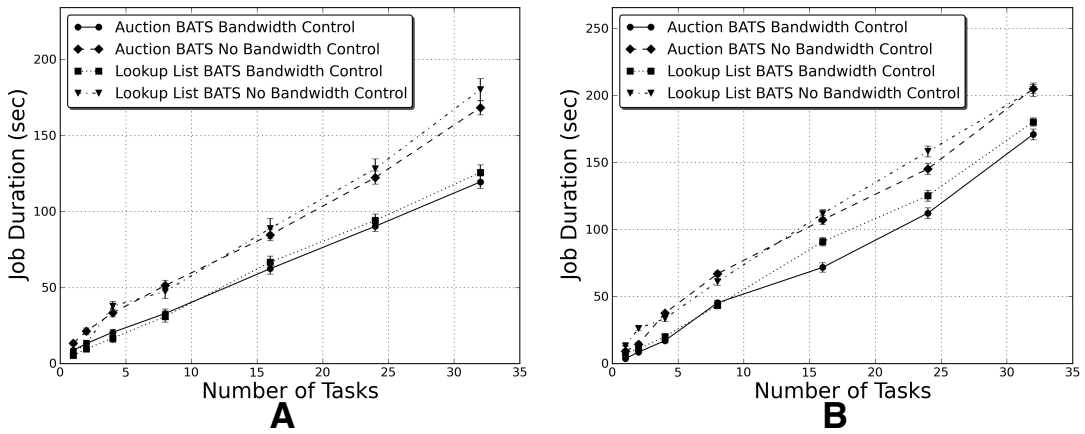


Figure 4.19: Auction and Lookup List comparison in Experiment A (left) and B (right).

The discovery phase, allowing each GA to discover and initialise its own neighbourhood of TEs by filling a lookup list, is performed only at the beginning of each job execution. This implies that, in spite of Lookup List updates, if the job is composed of many tasks, such lookup list could be very different from the real-time status of the TEs toward the end of the job completion. Thus, when the number of

tasks increases, this partial out-of-date stored information leads to the GA performing incorrect offload decisions. With very little additional overhead caused by the broadcast performed at the beginning of each task computation, the GA-initiated Auction algorithm always achieves better performance. In fact, while during every decision making phase the Auction algorithm deals with an up-to-date snapshot of the system reality in terms of load and network conditions, the Lookup List progressively worsens such snapshot, thus leading to erroneous offload decisions.

Consequently, since the Auction algorithm is more flexible and robust both regarding the number of tasks $Task_{i,j}$ required to be load distributed, and the number of the GAs simultaneously operating within the environment, it represents a better choice to apply the collaborative computing DWAG paradigm.

In *Experiment B*, we also compare the performance of both algorithms against a random task assignment. This implies that within both Auction and Lookup List algorithms the BATS heuristic is not applied and it is instead replaced by a random decision making process. This means that in both algorithms, for each decision making phase performed to load distribute a task execution, a TE is picked at random. This was done to test the approaches against a simple standard policy used for analysis in load sharing approaches [Eager et al., 1986]. As expected, the experimental results shown in Figures 4.18A and B indicate that our approach, which is capable of making informed decisions about where to distribute tasks, always achieves better performance than a pure random approach.

4.7 Summary of Results

We are now in a position to summarise the most interesting findings drawn from our empirical experimental study, as follows:

- For both algorithms Auction and Lookup List proposed to deal with load distribution through the DWAG paradigm, considerable performance improvements are achieved whenever a BATS heuristic, combining both TEs computational and local network conditions, is applied. Hence experimental results show that computationally intensive applications, required to be collaboratively executed, achieve faster completion times whenever informed decisions about heterogeneous traffic contention levels are taken into account before blindly dealing with task offload. This happens whenever homogeneous kind of tasks are distributed within the environment.

- As expected, the *job duration* of both algorithms decreases by increasing the number of the TEs acting in the environment up to a value representing the GA's parallelism. Moreover, the latency value increases by increasing the number of I/O bursts necessary to be performed to allow remote task computation (while keeping constant the amount of CPU bursts per task to be computed). Again, the *job duration* increases by increasing the number of tasks forming the computationally intensive job to be computed. This behavioural pattern recursively repeats for both algorithms not only when the TEs computational capabilities are accounted, but also when a combination of CPU and bandwidth information are coupled through BATS heuristic.

- Comparing the relative performance of the Auction and the Lookup List algorithms, it is possible to notice a tendency as follows. While the proactive more complex TE-initiated algorithm

(Lookup List) slightly outperforms the reactive GA-initiated one (Auction) whenever a single GA is exclusively entitled to distribute load in the system, the opposite behaviour is displayed when multiple GAs concurrently distribute tasks within a shared environment. This happens because when one GA exclusively offloads tasks to the TEs, it is the only entity able to actively change the TEs status and thus the GA's lookup list better approximates the actual load of the TEs. On the other hand, when several GAs simultaneously offload tasks into the environment, the load of each TE can be actively modified by the concurrent action of multiple GAs without them notifying each other the provoked load variations. This behaviour leads in the long run to progressively increase the gap between the information stored into the lookup lists at the GA's side and the actual TE load. Hence experimental evaluation with homogeneous task configurations shows that the very little overhead introduced by the Auction algorithm leads, in the long run, to statistical performance benefits. Thus, in the following evaluations we will focus our attention on the more flexible, robust and stable Auction algorithm.

4.8 Discussion

In this chapter, we presented the bandwidth problem as the impact of environmental network traffic contention on the distribution of computationally intensive applications according to the DWAG paradigm. In particular, we proposed two load sharing algorithms, the Auction and the Lookup List, chosen and adapted to be implemented in a completely distributed manner on deployed systems with the aim of applying job distribution. Moreover, we devised and integrated within the algorithmic decision making process a BATS heuristic combining both the local information regarding TE available computational resources and the local amount of existing network contention in the area where TE is located. Experimental results showed: (i) the crucial impact of existing network traffic on node collaborations occurring to achieve the completion of a computationally intensive application; (ii) the greater adaptability of the simple and robust Auction algorithm in making more informed decision based on updated network conditions.

Chapter 5

Impact on Distributing Heterogeneous Tasks

In Chapter 4, we described the bandwidth problem as the impact of environmental network traffic contention on the distribution of computationally intensive applications according to the DWAG paradigm. In particular, we presented two load sharing algorithms, the Auction and the Lookup List, chosen and adapted to be implemented in a completely distributed manner on deployed testbeds with the aim of applying job distribution. Moreover, we devised and integrated within the algorithmic decision making process a BATS heuristic combining TE's local available computational resources and local network conditions.

In Chapter 4, experimental results measuring the impact of TE local network contention on completing homogeneous applications, have shown: (i) the crucial impact of accounting for radio communication in collaboratively computing a job; (ii) the flexibility of the Auction algorithm in dealing with network conditions changes, thus resulting in prompt adaptations leading to informed decisions based on up-to-date network radio communication values. The aim of this chapter is to progress such empirical investigation by analysing the impact of distributing heterogeneous applications and tasks within the environment and by introducing a variation of heuristics to handle such distributions.

The remainder of this chapter is thus organised as follows: (i) in Section 5.1, we characterise a set of heterogeneous applications, differing from both a computational and a communication perspective; (ii) in Section 5.2, we devise an algorithm computing a theoretical lower-bound representing the view of an oracular system capable of taking, in each instant of time, the best offload decisions; (iii) in Section 5.3, we define an additional heuristic to measure the impact of distributing heterogeneous kinds of applications within the environment and we compare it against BATS and the theoretical lower-bound; (iv) in Section 5.4, we test the proposed DWAG paradigm by distributing the defined heterogeneous applications and applying the set of specified heuristics with actual computation, actual profiling of the medium, and actual network traffic for a multiplicity of settings; and (v) in Section 5.5, we summarise the experimental results.

5.1 Heterogeneous Application Types

We are now in a position to tackle an experimental analysis aiming at investigating the impacts of distributing heterogeneous kinds of applications within the environment. Our aim is again focused on

measuring the importance of taking network conditions into account while performing collaborative computation through DWAG collaborations, and thus achieving job completion by distributing each task $Task_{i,j}$, into which Job_i is split, from the GA to the auxiliary TE nodes.

In Chapter 4, the experimental evaluation highlighted that, in dealing with up-to-date traffic values, reactive approaches promptly detect and react to sudden environmental changes, thus leading to consistent good performance. Hence, for the experimental analysis undertaken in the current chapter, we focus our attention on the reactive GA-initiated Auction algorithm.

We are now interested in investigating the impacts of job, and consequently tasks, heterogeneity on achieving fast DWAG completion times. As illustrated in Chapter 4, the applications adopted so far within the experimentation were homogeneous. In fact, for each $Task_{i,j}$ forming Job_i the following conditions were satisfied: (i) the same number of messages were offloaded from the GAs to the TEs and uploaded back from the TEs to the GAs; and (ii) the same amount of computation was performed at the TE side. This thus translates into identical Input/Output (I/O) bursts and CPU cycles for each task computation.

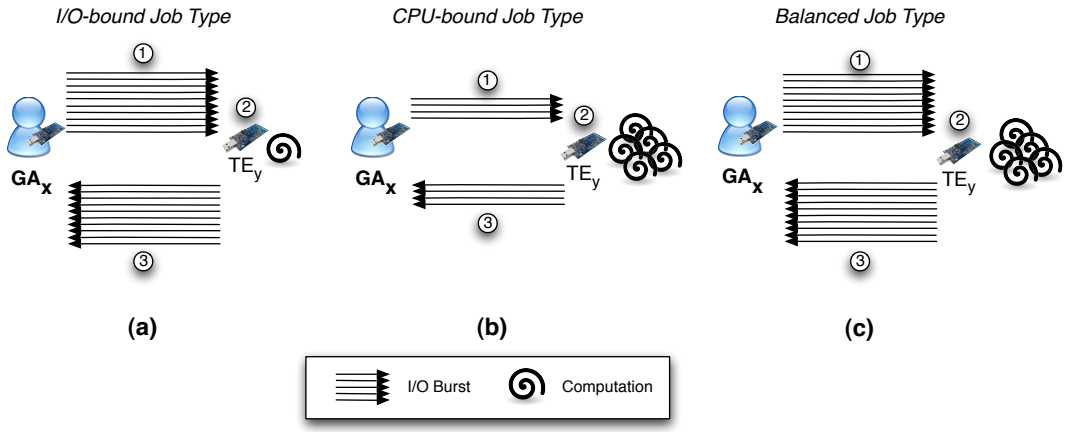


Figure 5.1: Different application types (I/O-, CPU-bound and balanced jobs).

Instead, we now define different sets of job classes, as follows:

- **I/O-Bound Job Type**

  As displayed in Figure 5.1(a), these applications are characterised by tasks with a considerable number of I/O bursts. Message streams are exchanged to allow both the GAs to supply the TEs with the quantity of information necessary to perform task remote computation, and the TEs to supply the GAs with the consequent upload of computed results. Moreover, I/O bursts also emulate inner communication exchanges required to progress with task execution. This kind of jobs is classified as I/O-bound since the actual amount of communication handled by the remote execution of each task has more impact than the number of CPU cycles performed on the TE side. Examples of I/O-bound applications include those involving target detection and tracking, node localisation, and all kind of jobs for which considerable communication is required to achieve overall completion.

- **CPU-Bound Job Type**

  As displayed in Figure 5.1(b), these applications are characterised by tasks with few I/O bursts and considerable remotely computed CPU cycles. Hence TEs, elected to compute CPU-bound tasks, spend on average most of their time performing computation with very few message exchanges. Thus, the corresponding jobs are classified as CPU-bound since the actual amount of computation involved within each remote task execution has more impact than the limited number of I/O bursts. Examples of CPU-bound applications include those for which sensor platforms are not only responsible for environmental data collection (e.g. habitat, wildlife, heating ventilation and air conditioning, glacier monitoring, etc.), but also for autonomous data compression, filtering and analysis. For this kind of applications, devices spend most of their time sensing and performing autonomous and local computations, periodically synchronising with each other only when updates and collaborations are required.

- **Balanced Job Type**

  Whenever applications are balanced, they are formed by tasks that are both I/O- and CPU-bound, thus demanding from both a computational and a communication perspective. As displayed in Figure 5.1(c), these applications combine both I/O-bound and CPU-bound properties, since tasks of which they are comprised necessitate both considerable I/O bursts and demanding CPU cycles to achieve completion. Examples of balanced applications include collaborative node localisation, emergency response data collection and compression, collaborative exploration of unknown environments, to name but a few. Hence applications where nodes are heavily required to communicate with each other to achieve job completion while, at the same time, they are also involved in considerably demanding computations.

We are now in a position to study the impact of accounting for network information on collaboratively distributing heterogeneous kinds of applications, namely I/O-bound, CPU-bound, and balanced, through DWAG. The issue that arises now concerns the way of heterogeneously characterising jobs in order for them to reflect possible representations of real-world applications.

In this dissertation, we heterogeneously characterise tasks, of which jobs are comprised, from a computational and a communication perspective by taking inspiration from the real-world task analysis undertaken within computational grid scenarios reported in the work of Raimondi et al. [Raimondi et al., 2008]. In their work, the authors present a method able to monitor timeliness, reliability and request throughput of web-service Service Level Agreements (SLAs) by translating such conditions into timed automata. However, their work has been of use for our purposes not because of the SLA method they use, but because of the analysis conducted on real-world data collected from a large-scale case study carried out for a service-oriented computational grid involving a chemistry application [Emmerich et al., 2005]. In this kind of computational grid application, a client component submits searches to a web service implemented as a BPEL workflow. The BPEL workflow calls a number of web services to submit different Fortran executables to compute resources, to visualise results and to upload the consolidated search result to a data portal. The authors monitored the BPEL workflows

generated by clients during 4 hours of activity. Approximately 16 batches of jobs were invoked, with the generation of nearly 230,000 SOAP messages. From the data collected, distributions for both task computation and communication emerged. In particular, the data reported [Raimondi et al., 2008] shows a Gaussian-like distribution to represent task I/O-bursts, and a Poissonian-like distribution to represent task CPU cycles.

Therefore, to cope with the lack of real-world data to characterise heterogeneous applications from both a computational and a communication perspective, we adopted the I/O and CPU distributions emerging from computational grid scenarios. Hence we applied the Gaussian distribution in Equation 5.1 to map the I/O burst and the Poissonian distribution in Equation 5.2 to map the CPU pattern behaviour. The values of σ and μ, for the Gaussian distribution, and λ, for the Poissonian distribution, have been tuned to represent significant I/O-bound, CPU-bound and balanced application type kind of settings. Section 5.4 provides a detailed description of the values selected for each distribution according to the specific application type. In the evaluation, we investigated a set of distributions (i.e. Binomial, Power) but since the results did not substantially differ from those obtained with the Gaussian and Poissonian functions, in this dissertation we decided to report exclusively those obtained with the latter.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.1}$$

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \tag{5.2}$$

Once the parameters for each distribution are set according to the type of application, the following actions are performed for each task computation: (i) a burst of I/O messages is delivered from the GA to the TE by randomly selecting a value within the Gaussian distribution; (ii) a burst of CPU cycles is performed by randomly choosing a value within the Poissonian distribution; (iii) a burst of I/O messages is uploaded back from the TE to the GA according to the value previously selected within the Gaussian distribution. This set of actions are repeated for each task computation $Repetition$ times (value kept fixed for each task), and thus $Repetition$ represents the number of repetitions undertaken by each task to achieve overall completion.

5.2 Lower-bound Computation

The aim of this chapter is to investigate the impact of network conditions in executing heterogeneous jobs. To cope with the lack of real data to describe computational and communication requirements of a variety of applications, we have taken inspiration from behaviours emerging from computational grids application and we have used them to characterise our application classes. This has been done by observing patterns from real data, by extracting approximations of distributions for both computational and communication task requirements and adapting values to WSNs real-time requirements.

In dealing with job heterogeneity, we are now interested in comparing our empirical findings against a theoretically estimated lower-bound task completion time computed by envisioning the presence of an oracular system capable of taking the best offload decision. Achieving a globally optimum task

scheduling is an NP-complete problem [Garey and Johnson, 1979], not only in a dynamic but also in a static system of nodes with an oracular overview of the global state of the system. The oracle is here assumed to have a full vision on the current task allocation, on the current load of the TE nodes and on the actual situation in terms of existing and generated traffic contention.

In order to obtain estimated task completion times comparable to those collected in our empirical results, we performed an experiment to determine for each kind of application approximate task duration values for both cases in which a new task is offloaded to a TE belonging to a congested or uncongested area. In particular, as illustrated in Figure 5.2, we positioned a single GA and a single TE in the environment, setting both their degree of parallelism to 1, thus ensuring that both nodes will deal with a single task at the time. Whenever a job is formed of multiple homogeneous tasks, the duration of a task is obtained by averaging the durations of all tasks forming that job. We then loaded the GA with different forms of application, each composed of 30 tasks characterised as described in Section 5.1. The values of σ, μ and λ were set for each application type, as described in Section 5.4, and maintained throughout the entire job execution.

Each task was thus offloaded and sequentially executed on the TE. We computed an approximate task duration value by averaging task duration values of all tasks composing each individual application. We repeated the experiment in situations both without (Figure 5.2(a)) and with (Figure 5.2(b)) an interferer node Str. In this way, we computed, to a good approximation, two values representing task duration $\Delta t_{Task_{i_{TE}}}$ of $Task_i$ if computed on a congested TE (i.e. cTE) or uncongested TE (i.e. ncTE). In the experiments, Str's radio transmission power was set to the second lowest level (i.e. -28 dBm in [Instruments, 2006]), in the same way as for the experimental evaluation in Chapter 4. Likewise Str was configured to send bursts of 3 bytes at regular intervals.

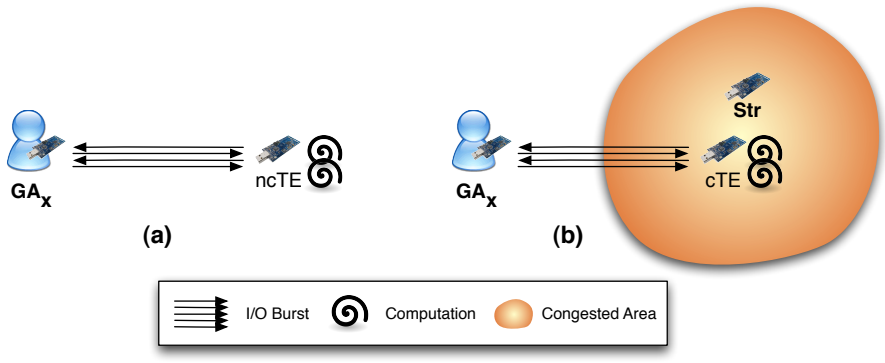


Figure 5.2: Experiment to compute $\Delta t_{Task_{i_{TE}}}$ task duration on a congested cTE or uncongested ncTE for each application type.

We now detail the algorithmic steps presented in Algorithm 5 responsible for lower-bound computation. The algorithm was implemented using Python 2.6.4 and run on a MacBook Pro 2.8GHz Intel Core Duo with 4GB RAM. In particular, the engine coding the main algorithmic logic was implemented by using the python Discrete Event Simulation [Müller, 2008, Müller, 2010] paradigm.

The algorithm is split into two parts: the first takes care of selecting the best TE (i.e. TE_{Best})

responsible for dealing with the computation of a task; the second one computes the actual *newTask*, thus it progressively updates the estimated task completion times according to the actual number of tasks already running on a TE node. Hence the first part describes what happens on the GA side, while the second describes what instead occurs on the TE side.

Let us now describe the meaning of the variables adopted in Algorithm 5. $\text{GA}_{busySlots}$ and $\text{TE}_{busySlots}$ represent the number of tasks actively running in each instant of time on the GA and the TE nodes, respectively. Such values are always required to be smaller than the maximum supported degree of parallelism, $\text{GA}_{maxSlots}$ and $\text{TE}_{maxSlots}$. In fact, the difference between the maximum number of tasks allowed to simultaneously run on a node (i.e. $\text{GA}_{maxSlots}$ and $\text{TE}_{maxSlots}$) and those actually running on it (i.e. $\text{GA}_{busySlots}$ and $\text{TE}_{busySlots}$) represents the number of free slots (i.e. $\text{GA}_{freeSlots}$ and $\text{TE}_{freeSlots}$), thus the number of tasks still potentially allocable on a node. In Algorithm 5, we indicate with $Task_i$ each one of the tasks already running on a node, maximum one for each slot, and with *newTask* the task entering the system and required to be allocated. TE_{thr} is an estimated value associated with each TE and computed by the GA whenever the latter is required to decide the best TE_{Best} node towards which to direct and allocate *newTask* computation. Other variables are used to represent either instants of time or task durations. For example, $t_{current}$ identifies the current instant of time, $t_{endTask_i}$ and $t_{endNewTask}$ are the estimated instants at which either a running task $Task_i$ or a new task *newTask* are envisioned to achieve completion. Similarly, Δt_{Task_i} and $\Delta t_{newTask}$ indicate task durations for $Task_i$ and *newTask*, respectively.

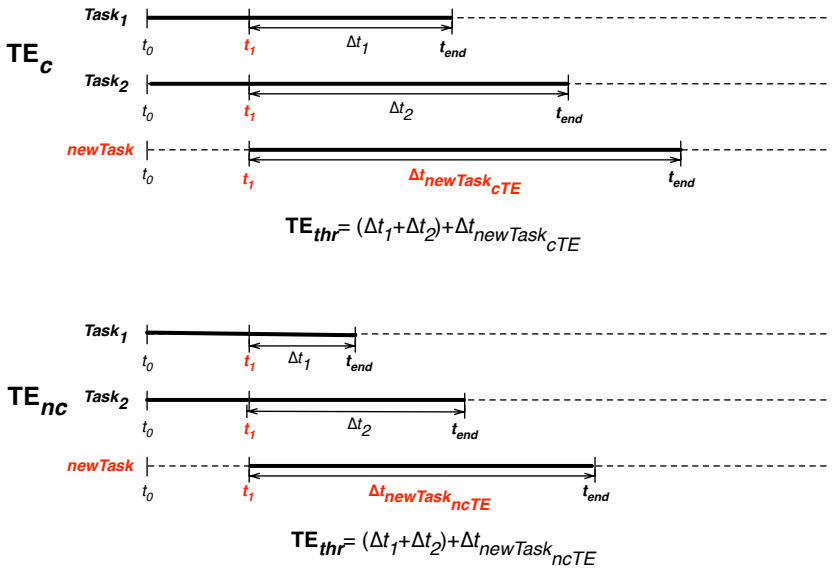


Figure 5.3: TE_{Best} selection within the theoretical lower-bound algorithm.

We are now in a position to describe the two parts composing Algorithm 5. The first part is performed at the GA side and concerns the selection of TE_{Best}. Thus, whenever the GA has available slots $\text{GA}_{freeSlots}$ to manage task allocation and whenever there is at least a TE similarly having at least a free slot $\text{TE}_{freeSlots}$ to handle a new task computation, the GA is required to decide onto which TE to distribute task computation. Hence the GA computes a TE_{thr} value for each TE. Such an estimated

threshold is computed by performing the sum of the differences between the $t_{endTask_i}$ estimated task completion time and the current instant of time $t_{current}$ for each running task $Task_i$ and further adding to such partially computed value the envisioned duration of *newTask* to be allocated. In particular, if *newTask* is forecast to be allocated on a cTE, thus belonging to a congested area, then its correspondent duration is set to $\Delta t_{newTask_{cTE}}$. On the other hand, if *newTask* is forecast to be allocated on a ncTE, thus belonging to an uncongested area, then its corresponding duration is set to $\Delta t_{newTask_{ncTE}}$. In particular, $\Delta t_{newTask_{cTE}}$ and $\Delta t_{newTask_{ncTE}}$ values are computed by performing the experiment described above. The TE with associated the minimum TE_{estThr} value is selected from the GA as the destination for *newTask*.

As illustrated in Figure 5.3, let us imagine having two TE nodes, cTE belonging to a more contended area and ncTE belonging to a less contended area, thus emulating the real situation of heterogeneous traffic congestion levels. Let us also assume that two tasks $Task_1$ and $Task_2$ are already running on each node thus cTE$_{busySlots}$ =ncTE$_{busySlots}$ = 2 and the full degree of parallelism is cTE$_{maxSlots}$ =ncTE$_{maxSlots}$ = 3. Whenever *newTask* enters the system at t_1, $Task_1$ and $Task_2$ have already been partially computed, thus it becomes crucial to measure the time still required by each of them to achieve completion, namely Δt_1 and Δt_2, respectively. Since the oracle is reckoned to know both $t_{endTask_1}$ and $t_{endTask_2}$, it computes the differences between $t_{endTask_1}$ and $t_{endTask_2}$ with respect to the current instant of time t_1. Moreover, the oracle is assumed to know the duration of *newTask* whether it is offloaded on congested cTE (i.e. $\Delta t_{newTask_{cTE}}$) or uncongested ncTE (i.e. $\Delta t_{newTask_{ncTE}}$). Thus, the threshold TE_{thr} is computed for each TE by adding together Δt_1, Δt_2 and $\Delta t_{newTask}$ values. The TE with minimum TE_{thr} is selected as targeted TE$_{Best}$ to handle *newTask* execution. In particular, since in Figure 5.3 ncTE is the least loaded node, it is therefore selected as target for distribution. Note that, since the algorithm runs in simulation, no task offload is actually performed.

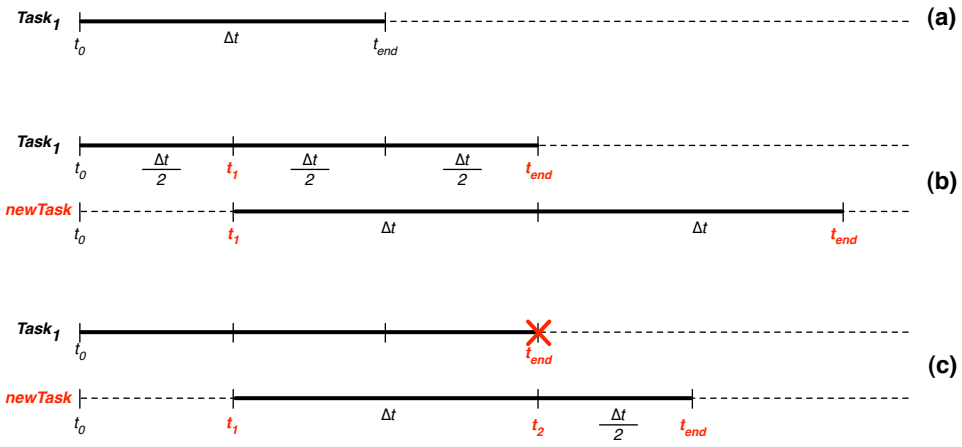


Figure 5.4: Task estimated completion time computation within the theoretical lower-bound algorithm.

The second part of Algorithm 5 takes care instead of the actual *newTask* computation and thus of the definition and update of the estimated completion time $t_{endTask_i}$ values of a task. In fact, whenever a new task *newTask* enters the system or one of those $Task_i$ already running on the TE completes its

execution, it is necessary both to set (for *newTask*) and to update (for $Task_i$) the estimated completion time for all tasks. In the former situation, in which *newTask* needs to start running on the previously selected TE_{Best}, for each $Task_i$ already running, it is necessary to undertake the following actions: (i) compute the remaining time $\Delta t_{remTask_i}$, for which running $Task_i$, multiplying the difference between the actual estimated completion time $t_{endTask_i}$ and the current instant $t_{current}$ for a number obtained by dividing the current number of running tasks incremented by 1 (i.e. $\text{TE}_{busySlots} + 1$) and the current one (i.e. $\text{TE}_{busySlots}$); (ii) compute the new $t_{endTask_i}$ of each $Task_i$ by adding to the current instant of time $t_{current}$ the new remaining task duration $\Delta t_{remTask_i}$. Moreover, *newTask* estimated completion time (i.e. $t_{endNewTask}$) is set by adding to the current instant of time $t_{current}$ a value computed by multiplying *newTask* duration $\Delta t_{newTask}$ by the actual number of running tasks incremented by 1. Since *newTask* is now allocated to the TE, the actual number of running tasks is incremented by 1 (i.e. $\text{TE}_{busySlots}++$), while the actual number of free slots, available to allocate task computation, is decremented by 1 (i.e. $\text{TE}_{freeSlots}--$).

The way in which this mechanism works can be easily explained with an example, as illustrated in Figure 5.4. Let us assume that a task $Task_1$ is already running on a TE (Figure 5.4(a)) when a new task *newTask* wishes to start its execution (Figure 5.4(b)). Moreover, let us assume that both $Task_1$ and *newTask* have the same Δt duration whenever each one of them is computed in an exclusive way on a TE node. This implies that, whenever either $Task_1$ or *newTask* runs on the TE, task duration is Δt. On the other hand, whenever both $Task_1$ and *newTask* are required to be simultaneously executed on a node, the task duration changes. In particular, according to the traditional parallel computing principles [Grama et al., 2003], a process execution proportionally increases or decreases according to the number of processes running in parallel. For example, in Figure 5.4(b) *newTask* starts when half of $Task_1$ (i.e. $\Delta t/2$) has already been computed. Hence $Task_1$ needs to double its remaining duration $\Delta t/2$ in order to achieve completion, thus its estimated completion time is updated to $t_{endTask_1} = t_0 + 3\Delta t/2$. Similarly, *newTask* updates its duration Δt to $2\Delta t$ and the correspondent estimated completion time to $t_{endNewTask} = t_1 + 2\Delta t$. The same procedure is repeated in reverse when a task terminates. For example, as illustrated in Figure 5.4(c), when $Task_1$ terminates at instant t_2, *newTask* no longer shares TE computational resources with any other task running in parallel. Hence its remaining duration is halved since it can now run twice as fast as before. Thus, *newTask* updates both its duration, previously set to $2\Delta t$, to $3\Delta t/2$ and its estimated completion time to $t_{endNewTask} = t_1 + 3\Delta t/2$. This situation is described in the second part of Algorithm 5. The actions to be undertaken are as above with the only difference being that the duration $\Delta t_{remTask_i}$ of each $Task_i$ is now obtained by multiplying the current value by a number obtained by dividing the current number of running tasks decremented by 1 (i.e. $\text{TE}_{busySlots} - 1$) with the current number of running tasks (i.e. $\text{TE}_{busySlots}$).

Recall that the computed values represent only a theoretical lower-bound obtained by making use of an empirical approximation of task duration (i.e. $\Delta t_{Task_{i_{cTE}}}$ and $\Delta t_{Task_{i_{ncTE}}}$ whether $Task_i$ is computed on a congested cTE or uncongested ncTE), but totally discarding any radio communication issue in terms of interference, collisions, retransmission mechanisms and delays.

Algorithm 5: newTask Computation Algorithm

```
/* GA_freeSlots/TE_freeSlots are number of GA/TE free slots.          */
/* GA_busySlots/TE_busySlots are number of GA/TE allocated slots.     */
/* newTask is a new Task to be allocated.                             */
/* Task_i are tasks running on TE, one for available slot.            */
/* TE_thr is an estimated threshold associated to each TE.            */
/* t_current is the current instant of time.                          */
/* t_endTask is estimated task ending time.                           */
/* Δt is task duration.                                               */
/* GA (Selection of TE_Best):                                         */
foreach (GA) do
    if (GA_freeSlots > 0) then
        foreach (TE) do
            if (TE_freeSlots > 0) then
                if (cTE) then
                    TE_thr = [ Σ_{i=1}^{TE_busySlots} (t_endTask_i − t_current)] + Δt_newTask_cTE;
                end
                else if (ncTE) then
                    TE_thr = [ Σ_{i=1}^{TE_busySlots} (t_endTask_i − t_current)] + Δt_newTask_ncTE;
                end
            end
        end
        GA selects TE_Best with minimum TE_thr to offloads newTask;
    end
end
/* TE (newTask Computation of t_endNewTask):                          */
foreach (event taskStart or taskFinish) do
    if (event taskStart) then
        foreach (TE_busySlots_i) do
            Δt_remTask_i = (t_endTask_i − t_current) · [(TE_busySlots + 1)/TE_busySlots];
            t_endTask_i = t_current + Δt_remTask_i;
        end
        t_endNewTask = t_current + Δt_newTask · (TE_busySlots + 1);
        TE_busySlots + +;
        TE_freeSlots − −;
    end
    if (event taskFinish) then
        foreach (TE_busySlots_i) do
            Δt_remTask_i = (t_endTask_i − t_current) · [(TE_busySlots − 1)/TE_busySlots];
            t_endTask_i = t_current + Δt_remTask_i;
        end
        TE_busySlots − −;
        TE_freeSlots + +;
    end
end
```

Finally, it is interesting to point out that in Algorithm 5 the oracle allocates tasks on the TEs reasoning on task estimated completion time thresholds and not on the computational load of a node. The reason for this choice is easily explained with the following example. Let us assume that a GA is entitled to distribute a task by choosing among two possible TEs, namely TE_1 and TE_2, each of which has only one slot available for task computation. Independently of TEs' local level of network traffic, if the offload decision is based exclusively on the TE remaining CPU load, then both TE_1 and TE_2 have the same remaining CPU load (i.e. in this case equal to one), hence the GA could independently select any one of the possible TEs. However, if the offload decision is instead based on the evaluation of tasks estimated completion time values and tasks independently running on TE_1 and TE_2 have different thresholds, then the GA would rather choose the TE with the smallest associated values. In fact, the smaller the estimated completion time values of the tasks running on a node, the sooner a new running task would be able to benefit from TE's full CPU capacity since it will not be required to concurrently share TE's resources with other tasks for some time, as long as those running reach completion soon.

5.3 Heuristic Set

In this section, we define the set of heuristics adopted to measure the impact of distributing heterogeneous applications within the environment. In particular, we empirically compare results achieved by adopting a novel heuristic against those obtained by applying the BATS mechanism presented in Section 4.3. Once again, the main aim is that of investigating the impact of local network congestion in making informed decisions about load distribution.

Recall that in Section 5.1, we have characterised a set of applications in terms of I/O-bound, CPU-bound, and balanced. I/O and CPU characterisations have been driven by adopting task distributions emerging from computational grid fields and adapting them to WSNs. Moreover, in Section 5.2 we described an algorithm to compute a theoretical lower-bound against which we plan to compare the collected experimental results.

We are now in a position to list the set of heuristics adopted in the evaluation undertaken in the current chapter.

- **BATS:**

  The Bandwidth-Aware Task Scheduling mechanism is applied in this context in the same way as explained in Section 4.3. Hence BATS heuristic is built by linearly combining the information about the available computational resources of a TE_y node and the local amount of network contention performed in the area where TE_y is located. In particular, for each TE_y node belonging to GA's neighbourhood dynamically discovered through DWAG, a score value $Score_{TE_y}$ is obtained by computing a linearly weighted score function combining computational $TE_{y(a_{CPU})}$ and communication $TE_{y(a_{Band})}$ availabilities of TE_y. The TE_{Best} with the highest computed score $Score_{TE_{Best}}$ is selected to handle the remote $Task_{i,j}$ offload.

- **Profile-Bandwidth-Aware Task Scheduling (P-BATS):**

  The introduction of heterogeneous job categories drove the definition of another heuristic com-

bining TE_y computational and communication capabilities with task characterisations. Thus, in order to make an informed decision about to which TE_y to offload task computation, the GA and the TE are required to undertake the computations described in Equation 5.3 (GA), and Equations 5.4 and 5.5 (TE), respectively. In particular, each TE_y computes and delivers to the GA two values. The first value $TE_{(loadCPU)}$ (Equation 5.4) measures the amount of remaining computation, and thus CPU cycles, still required to be performed at TE_y side. This value is computed by multiplying, for each task already running on TE (i.e. $TE_{busySlots}$), the number of remaining repetitions $Task_{i_{remRpt}}$ with the remaining number of CPU computations $Task_{i_{remCPU}}$ to be undertaken (i.e. $Task_{i_{remCPUCycle}} = Task_{i_{remRpt}} \cdot Task_{i_{remCPU}}$). The partial values are then added together to form $TE_{(loadCPU)}$. Similarly, the second value $TE_{(loadI/O)}$ (Equation 5.5) measures the amount of remaining communication, and thus I/O bursts, still needed at TE_y, and thus the amount of communication affecting the local network conditions in which TE_y is located. Also this value is computed by multiplying, for each task already running on TE (i.e. $TE_{busySlots}$), the number of remaining repetitions $Task_{i_{remRpt}}$ and the number of remaining I/O bursts $Task_{i_{remI/OBurst}}$ to be sent (i.e. $Task_{i_{remI/OBurst}} = Task_{i_{remRpt}} \cdot Task_{i_{remI/O}}$). The partial values are then added together and the value obtained is multiplied by the existing local traffic contention $TE_{y(o_{Band})}$, identifying the level of bandwidth occupancy in TE_y's surrounding area, to form $TE_{(loadI/O)}$. This is in accordance with the task characterisation presented in Section 5.1, for which each task is characterised by the repetition of the actions (i) burst of offload messages, (ii) computation, and (iii) burst of upload messages, with the values for the bursts randomly picked within the distributions. Once $TE_{(loadCPU)}$ and $TE_{(loadI/O)}$ are delivered from each TE_y forming DWAG to the GA through the selected load distribution algorithm, the GA then associates with each TE_y a score value $Score_{TE_y}$ (Equation 5.3) by computing a linearly weighted score function combining $TE_{(loadCPU)}$ and $TE_{(loadI/O)}$ with information about the profile characterisation of the new task *newTask* to be offloaded. Thus, $newTask_{loadCPU}$ is the value of the computational load associated with *newTask* and computed according to *newTask* profile description $newTask_{loadCPU} = newTask_{rpt} \cdot newTask_{CPU}$. Similarly, $newTask_{loadI/O}$ represents the communication load added to the system whenever offloading *newTask*, again computed according to the profile characterisation $newTask_{loadI/O} = newTask_{rpt} \cdot newTask_{I/O}$). The TE_{Best} with the minimum computed score $Score_{TE_{Best}}$ is selected to handle the remote $Task_{i,j}$ offload. Note that, in order for this heuristic to be applied, it is necessary not only to be aware of TE local network information, but also of the task characterisation provided by the profiler. As stated in Section 1.4, we make the assumption of having such information available within the current dissertation.

GA Calculations:

$$Score_{TE_y} = (weight_{CPU} \cdot TE_{y(loadCPU)} \cdot newTask_{loadCPU}) + \quad (5.3)$$

$$+ \quad (weight_{Band} \cdot TE_{y(loadI/O)} \cdot newTask_{loadI/O})$$

TE Calculations:

$$TE_{(loadCPU)} = \sum_{i=1}^{TE_{busySlots}} Task_{i_{remCPUCycle}} \tag{5.4}$$

$$TE_{(loadI/O)} = (\sum_{i=1}^{TE_{busySlots}} Task_{i_{remI/OBurst}}) \cdot TE_{y(o_{Band})} \tag{5.5}$$

Finally, recall that for the current experimental evaluation we adopt the same load distribution algorithms as described in Section 4.2. In particular, since the experimental evaluation carried on in Chapter 4 has shown that the small amount of overhead introduced by the Auction brings considerable benefits in handling offload decisions, so we focus the current evaluation mainly on the Auction algorithm.

5.4 Experimental Evaluation

We now have the information necessary to detail the set of experiments undertaken within deployed systems to measure the impact of network information on distributing heterogeneous application categories within the environment, while introducing a variation of heuristics.

Recall that the results presented in this dissertation report information gathered from experimentation on deployed testbed. Moreover, we evaluate our work according to the assumptions presented in Section 1.4 and the experimental criteria stated in Section 1.5. As described in Section 1.3, we drive our evaluation along the latency dimension since latency translates into the amount of delay brought by radio communication links whenever diverse offload decisions are undertaken. Among the multitude of performed experiments, we decided to report in this dissertation the following findings since they are those corresponding to significant data variations.

In the experiments, we adopt the following metrics:

- *Job Duration:*

  This metric measures the overall duration of Job_i from the beginning to the end. Thus, this is the time spent to allow Job_i to complete the totality of tasks $Task_{i,j}$, into which it has been split.

- *Task Duration:*

  This metric measures the duration of each task $Task_{i,j}$ computed. Each task is represented by its $Task_{ID}$.

- *Task Completion Time:*

  This metric measures the instants of time in which each task $Task_{i,j}$, into which Job_i is split, progressively reaches completion. Each task is represented by its $Task_{ID}$.

The algorithms were run 30 times, more if necessary to obtain statistically valid results. Each point in the graphs represents the average of all runs and the error bars show the standard deviation over the

runs. In each experiment, we vary the values of one input parameter and assign the default values to the others if not mentioned otherwise.

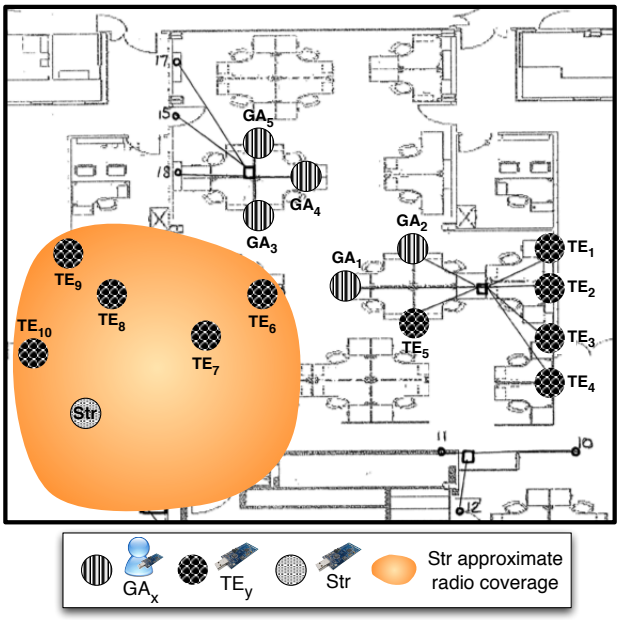


Figure 5.5: Map of UCL-CS HEN device deployment for job heterogeneity experiments.

The experiments are again performed on the UCL-CS HEN TMote Sky testbed, whose details were provided in Section 4.6.2. The roles for the GA, the TE and the Str nodes were assigned as illustrated in Figure 5.5.

In a similar manner to homogeneous experimentation, the degree of parallelism supported by both GA and TE nodes is 3. Again, we set the value to 3 since up to this value the context switching among processes does not affect the computational resources of TMote Sky devices.

In the following experiments, communication is handled through the Rime [Dunkels et al., 2007] protocol stack implemented within the Contiki OS. As illustrated in Section 2.2.3, Rime was devised to cope with increasing (i) network heterogeneity, (ii) number of link layers, (iii) MAC protocols, and (iv) underlying transportation mechanisms. Hence, since the experiments presented in this chapter aim at dealing with multiple heterogeneous network traffic and profile characterisation, we decided to test our algorithms by adopting the Rime stack instead of uIP TCP/IP used for homogeneous experimentation in Chapter 4.

Like the experiments in Chapter 4, we programmed TMote Sky devices by setting the Radio Frequency (RF) ZigBee channel to 26 to minimise interference.

In order to generate heterogeneous environmental network contention traffic, while allowing experimental repeatability, we make again use of an interferer node Str locating it within the radio range of only a subset of TE nodes. We adopt for Str the same message transmission rate as that used within the homogeneous scenario in Section 4.6.1. Moreover, we again set Str's radio transmission power to the second lowest possible level and inject bursts of 3 bytes into the network after a computation is executed. This choice was made to avoid saturating completely TEs network bandwidth in their reception areas.

Moreover, both the GAs and the TEs radio transmission power has been kept to the original Contiki OS default value (i.e. 0 dBm in [Instruments, 2006]) to allow all nodes to be in range of one another.

For the set of experiments presented in this chapter, we use the classes of applications defined in Section 5.1: I/O-bound, CPU-bound, and balanced. In each experiment, the offload/upload I/O bursts and the CPU cycles vary according to the specific Gaussian and Poissonian distributions. Details about the settings for the distribution parameters and the $Repetition$ value are provided within each of the following experimental setups. Moreover, we apply the set of heuristics described in Section 5.3 to the DWAG paradigm and compare the experimental results against a theoretical lower-bound computed as described in Section 5.2.

In order to apply the heuristics described in Section 5.3, it is necessary to compute $weight_{CPU}$ and $weight_{Band}$. For the BATS heuristic, these values are computed as described in Section 4.6.1 and thus weights $weight_{CPU}$ and $weight_{Band}$ are tuned to 16 and 1, respectively. Analogously, for the P-BATS heuristic weights $weight_{CPU}$ and $weight_{Band}$ are differently tuned for each of the experiments, following the same procedure as that adopted for BATS. Values of $weight_{CPU}$ and $weight_{Band}$ are detailed in each of the experiments.

Recall that, in the following experiments, we adopt the load distribution algorithms as described in Section 4.2. Hence we compare the performance of the Auction algorithm analysing the impact of the network conditions in dealing with traffic contention during distribution by adopting the heuristics in Section 5.3. In particular, we collect and analyse data coming from situations as follows:

- ***Heuristic with No Bandwidth Control*** **(**$weight_{CPU} \neq 0$ ***and*** $weight_{Band} = 0$**):**
  In this situation, we introduce into the system the interferer node Str in order to generate additional network traffic. However, whenever distribution is required, the decision is taken based only on TEs computational capabilities. This situation translates into distributing load while ignoring the local network traffic information, and thus exclusively accounting for TE computational availability, hence setting $weight_{CPU} \neq 0$ and $weight_{Band} = 0$ in each of the heuristics presented in Section 5.3.

- ***Heuristic with Bandwidth Control*** **(**$weight_{CPU} \neq 0$ ***and*** $weight_{Band} \neq 0$**):**
  In this situation, the system is again affected by the presence of the interferer node Str responsible for injecting within the environment heterogeneous amounts of traffic contention. However, the offload decision accounts here for the evaluation of both factors: TEs local computational and network availabilities. This situation translates into distributing under heavy communication load, while combining information regarding both TE local network conditions and internal computational availabilities, hence setting $weight_{CPU} \neq 0$ and $weight_{Band} \neq 0$ in each of the heuristics presented in Section 5.3.

In these experiments dealing with heterogeneity, we also report a statistical analysis conducted by applying One-Way Anova with Replication [Field and Hole, 2008] to the experimental values. The statistical modelling technique called Analysis of Variance (Anova) was chosen because it is designed to

test the statistical difference among two or more independent groups when data collected is normal and homoscedastic. In particular, One-Way Anova with Replication is used when three or more experimental groups (i.e. three or more means) are compared and the same participants are used in each group. The null hypothesis to be rejected each time is that the averages of all measured experimental sets are the same. According to the Anova technique, experimental results have a statistical impact when $p < .05$.

In each of the following sections, we detail the experimental setup and data analysis. In particular, in Section 5.4.1 we report a brief comparison between the Auction and the Lookup List algorithms again strengthening our decision to focus mainly on the Auction algorithm for the further experimental evaluation. In Section 5.4.2 we present results deriving from the GAs distributing I/O-bound jobs. In Section 5.4.3, the GAs deal with CPU-bound jobs, while in Section 5.4.4, the GAs cope with balanced jobs. In Section 5.4.5, we describe a summarising experiment in which different GAs simultaneously distribute a mixture of application categories within the environment. For each experimental set, we measure the system performance for a multiplicity of node configurations by varying the number of the GAs and the TEs actively working within the environment.

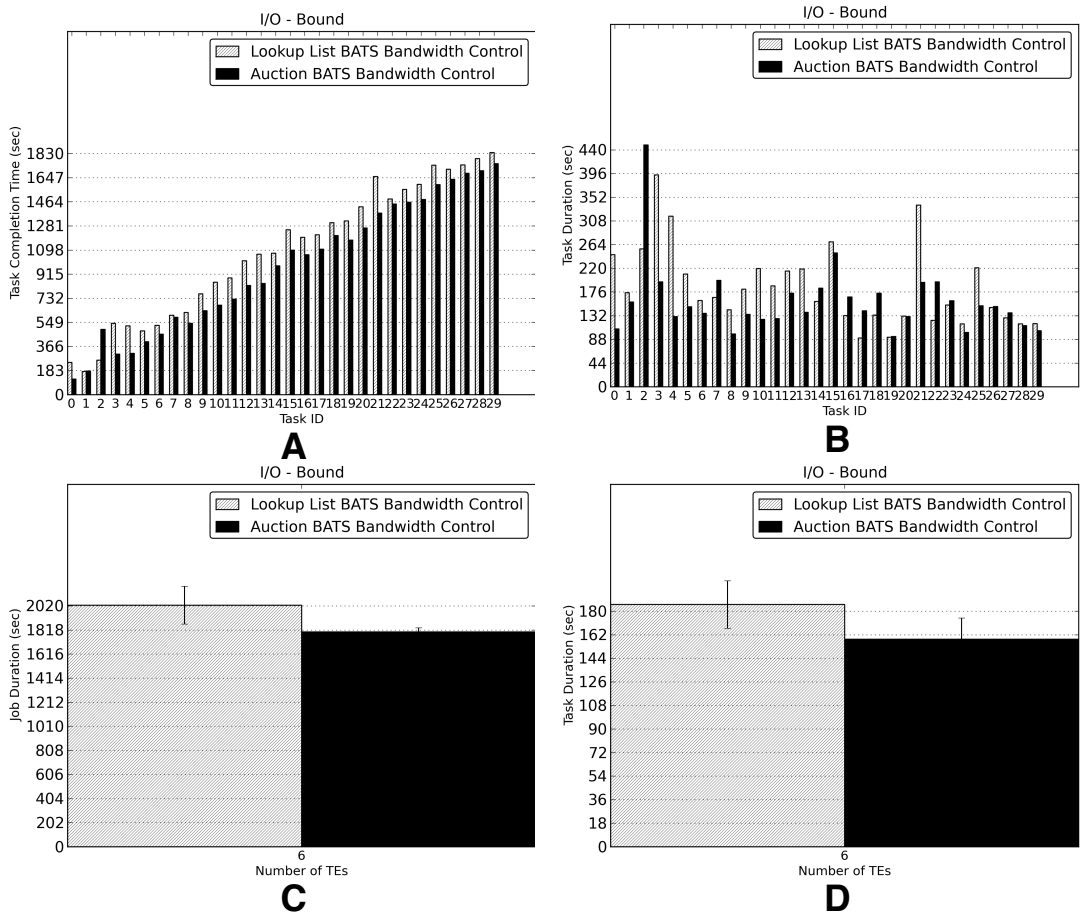


Figure 5.6: Performance comparison for Auction and Lookup List with I/O-bound jobs.

5.4.1 1^{st} Experimental Setup and Results (Auction vs. Lookup List)

The aim of this first set of experiments is to report a brief comparison between the Auction and the Lookup List algorithms. In particular, for the following experimental setup, we adopted 4 GAs and 6 TEs. Moreover, we decided to perform the comparison running the experiments with I/O-bound and balanced jobs, since they are more likely to be affected by local network information and thus bandwidth control mechanisms are likely to have a greater impact. In particular, the GAs firstly offload I/O-bound and then balanced jobs. Jobs and tasks have been configured according to the I/O-bound (Section 5.4.2) and balanced (Section 5.4.4) settings described in details in the following sections. Each job has been split into 30 tasks and each task is formed by 30 repetitions of the actions (i) burst of offload messages, (ii) computation cycles, and (iii) burst of upload messages.

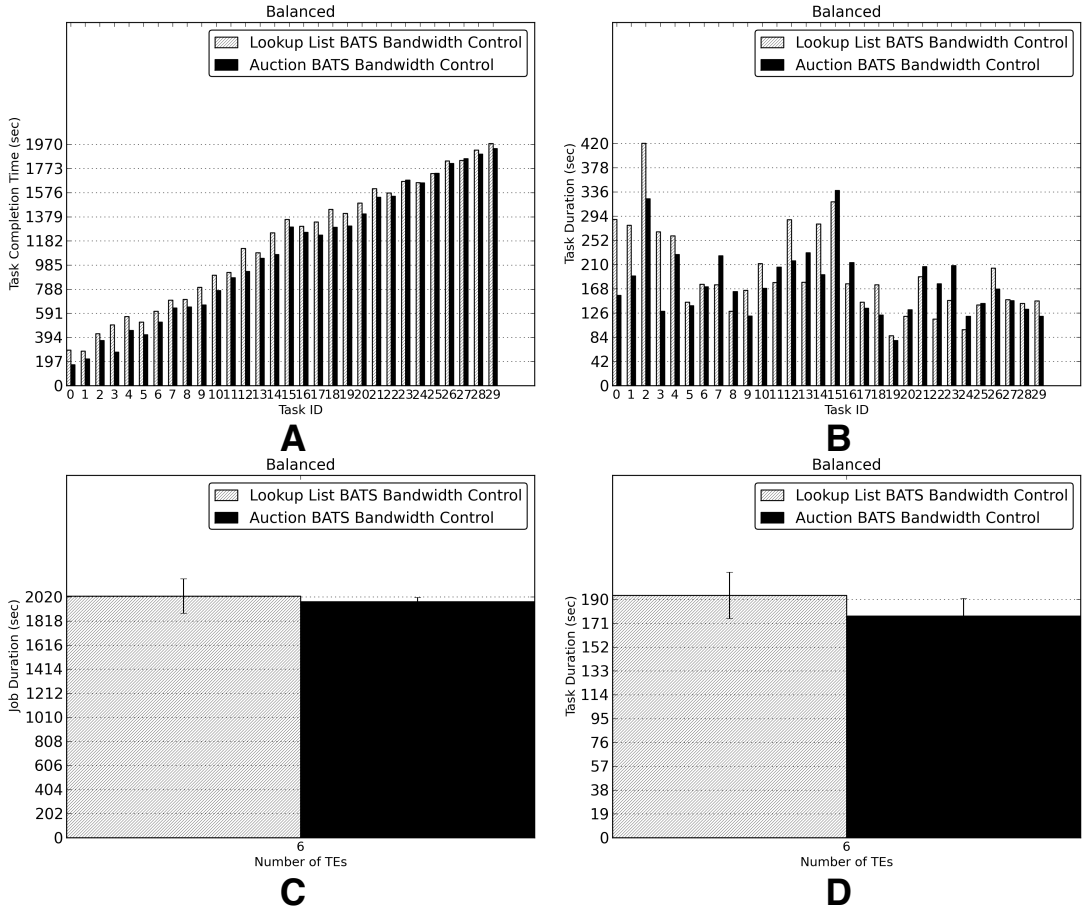


Figure 5.7: Performance comparison for Auction and Lookup List with balanced jobs.

In Chapter 4, we concluded the experimental analysis by stating that, the Auction algorithm outperforms Lookup List because of its ability to make more informed decisions based on up-to-date TEs status. As illustrated in Figures 5.6 and 5.7, by running both Auction and Lookup List with BATS heuristic and bandwidth control mechanism, we confirm the trend of Auction outperforming Lookup List. This is displayed for the experiments with I/O-bound jobs for *task completion time*, *job duration* and *task duration* (Figure 5.6). Whenever multiple GAs instead distribute cumbersome balanced tasks

I/O-bound job setup	
Number GAs	5
Number TEs	10
Number Tasks	30
Repetitions	30
Gaussian σ	5
Gaussian μ	30
Poissonian λ	2

Table 5.1: Experimental setup for I/O-bound job experimentation.

on a fixed number of the TEs, then the improvement is partially obfuscated (Figure 5.7). This behaviour is also confirmed by the computation of Anova. Regarding the Anova analysis, the following results have been obtained: [F(1,30)=1.3785, p=0.2496] in Figure 5.6C; [F(1,58)=2.154, p=0.1476] in Figure 5.6D; [F(1,22)=0.0065, p=0.9363] in Figure 5.7C; [F(1,58)=0.9096, p=0.3442] in Figure 5.7D. In an average case, however, the Auction algorithm would offer a better alternative for distribution than Lookup List. For this reason, we focus the following experimental analysis exclusively on the Auction algorithm.

5.4.2 2^{nd} Experimental Setup and Results (I/O-Bound Jobs)

The aim of this second set of experiments is the investigation of the impact of network conditions in dealing with the distribution of I/O-bound applications through DWAG collaborations. The experimental setup is as described in Table 5.1, with 5 GAs and 10 TEs. The former are pre-loaded with I/O-bound jobs each of which is split into 30 tasks. Each task is formed by 30 repetitions of the actions (i) burst of offload messages, (ii) computation cycles, and (iii) burst of upload messages.

The I/O burst and computation cycle values are randomly chosen within the Gaussian (communication) and Poissonian (computation) distributions set to map I/O-bound applications, as illustrated in Section 5.1. The values, randomly picked within the distributions, remain constant all over the task execution for each repeated cycle, but they are different for every task. Thus, once an I/O burst value is selected, it is used both for the task offload and upload. Hence, once the set of values for all tasks has been selected, the same sequence is used to allow experimental repeatability. For example, let us assume that we have an I/O-bound job, formed by 30 I/O-bound tasks. In order for each task to achieve completion, the following actions need to be undertaken: (i) two values X and Y are randomly selected from the Gaussian and Poissonian distributions, respectively; (ii) for every repeated cycle, an offload burst of X messages is sent from the GA to the TE, Y computational cycles are computed on the TE, and an upload burst of X messages is delivered back from the TE to the GA. Each offload message contains a value, which is then used by the TE to perform the computation of a function Y number of times.

The values adopted within the distributions to characterise I/O-bound jobs are as follows: $\sigma = 5$ and $\mu = 30$ for the Gaussian distribution, $\lambda = 2$ for the Poissonian distribution. The values of the weights within P-BATS have been set to $weight_{Band} = 1$ and $weight_{CPU} = 11250$ for I/O-bound jobs

and they have been tuned adopting the same procedure as that used for BATS but applied to P-BATS. Experimental parameters were tuned with such experimental values in order to create range variations compatible with the settings chosen for the experiments with homogeneous tasks in Chapter 4.

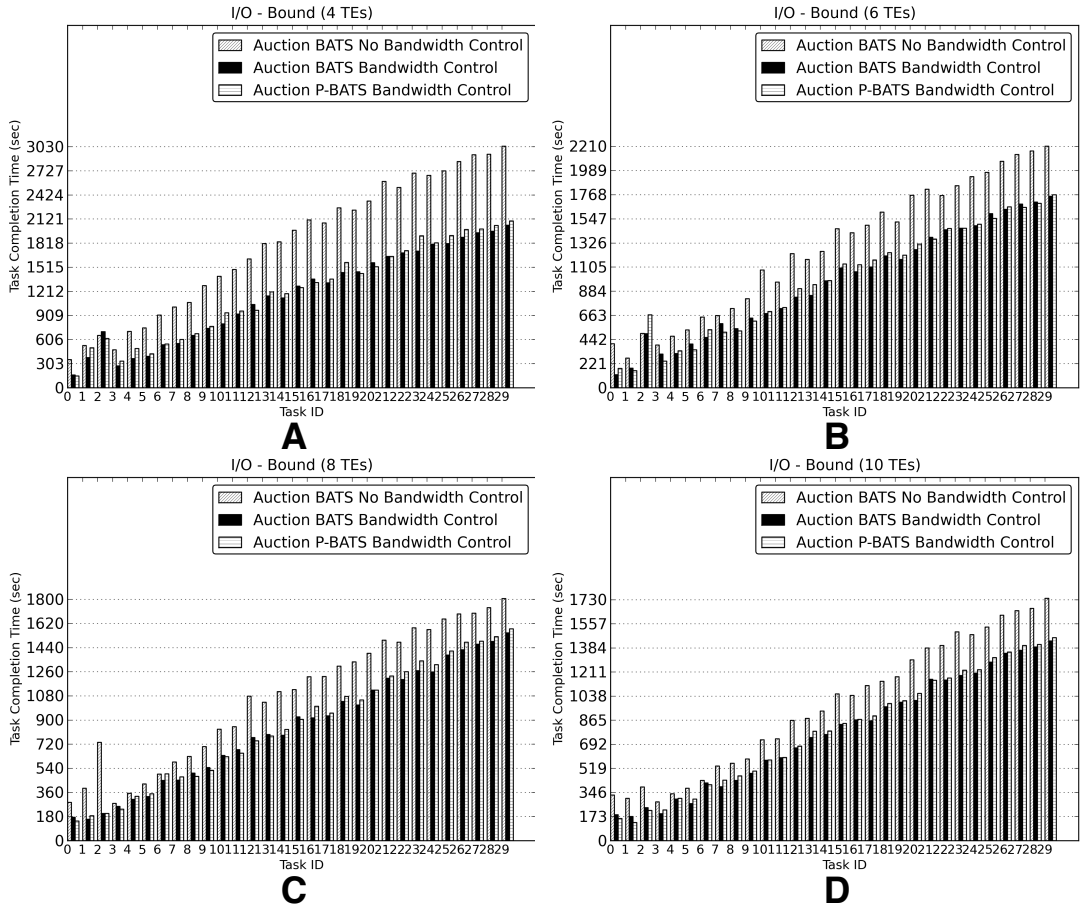


Figure 5.8: Effects of varying TE and $Task_{ID}$ on task completion time in I/O-bound jobs.

Effect of varying the number of the TEs: The aim of this experiment is to show the nature of the speed-up obtained if the number of the TEs available for computation in the system increases.

We vary the number of the TEs from 4 to 10, progressively adding 2 TEs at a time and locating each of them in the two diversified areas of influence: one affected by Str additional network contention, the other free from Str influences. In this experiment, 4 GAs simultaneously offload I/O-bound jobs each of which is composed of heterogeneous tasks profiled according to the distribution values detailed in Table 5.1. The number of the GAs is kept fixed within the experiment.

As illustrated in Figure 5.8, experimental results show that in terms of *task completion time* both BATS and P-BATS heuristics with congestion control significantly outperform the case in which only TEs computational requirements are accounted. This pattern is detected for each TE variation (Figure 5.8A-B-C-D), displaying a consistent trend. In addition, the increasing TE number reduces the performance gap between the two cases with and without bandwidth control for both heuristic mechanisms. This is as expected, since the computational load to be distributed remains constant, despite the

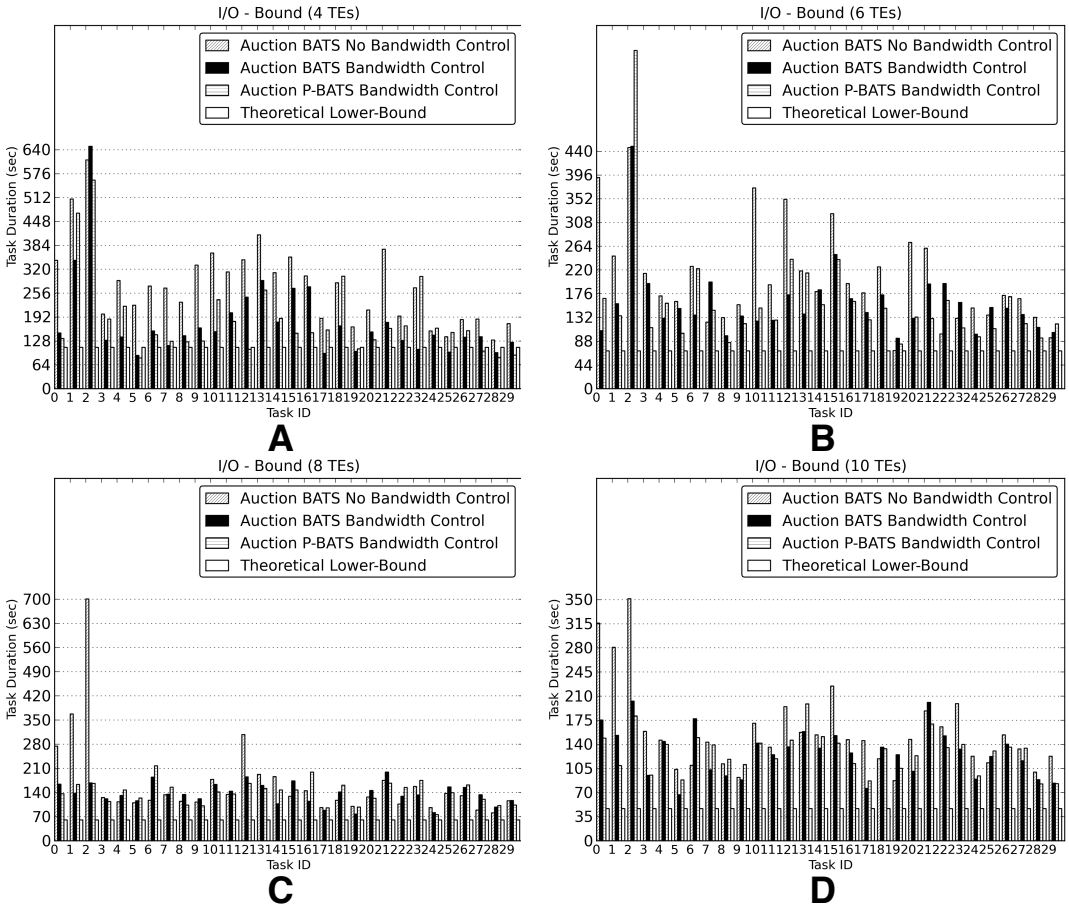


Figure 5.9: Effects of varying TE and $Task_{ID}$ on task duration in I/O-bound jobs.

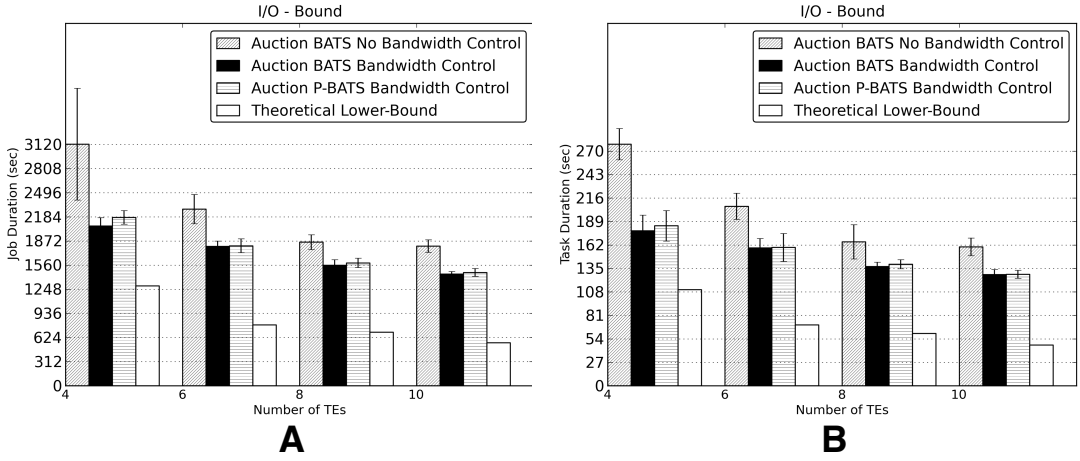


Figure 5.10: Effects of varying TE on job duration and task duration in I/O-bound jobs.

Number TEs	F-value	p-value
4 TEs	F(2,57)=9.371	p=0.003029
6 TEs	F(2,57)=21.99	p=8.351e-08
8 TEs	F(2,81)=34.935	p=1.149e-11
10 TEs	F(2,57)=56.05	p=3.471e-14

Table 5.2: Anova analysis for job duration varying TE in I/O-bound jobs (Figure 5.10A).

Number TEs	F-value	p-value
4 TEs	F(3,116)=16.067	p=8.435e-09
6 TEs	F(3,116)=17.339	p=2.282e-09
8 TEs	F(3,116)=15.192	p=2.107e-08
10 TEs	F(3,116)=46.727	p=2.2e-16

Table 5.3: Anova analysis for task duration varying TE in I/O-bound jobs (Figure 5.10B).

increasing TE number. Thus, the more TEs resources become scarce, the more the overall *task completion time* increases (increased gap between with and without bandwidth control cases in Figure 5.8A). On the other hand, whenever additional TEs resources suddenly become available to distribute the same amount of load, it is possible to notice performance improvements (decreased gap between with and without bandwidth control cases in Figure 5.8D).

The unusual behaviour of task 2 and 3 in Figures 5.8A-B is explained as follows. Whenever all the GAs simultaneously start offloading, the TEs find themselves dealing with a sudden burst of requests. Thus, the least congested TEs, being the primary target for distribution, quickly saturate their computational slots (actual degree of parallelism supported by each device) and some of the tasks have no choice but to be offloaded towards more congested TEs. This leads to a peak of the *task completion time* for the first tasks to be computed which then quickly normalises as the experiment progresses. It can, in fact, be noticed that the unusual delay caused by the initial burst of requests terminates approximately at the same time as task number 10 (Figure 5.8A) is offloaded without compromising the *task completion time* or times in which the experiment ends. The same behaviour is less obvious for Figures 5.8C-D since, in such cases, the available TEs are sufficient to handle the initial burst of requests.

As illustrated in Figure 5.9, considerations similar to those regarding *task completion time* can be propagated to *task duration*. In particular, as the graphs show, sometimes the *task duration* for heuristics with bandwidth control displays increased values with respect to the case where bandwidth control is disregarded. This happens because the mechanism distributing load exclusively accounting for TEs computational capabilities does not exclusively offload tasks onto congested TEs. In fact, it might happen that a node more capable from a computational perspective is also located within a uncongested area. Thus, the overlap of conditions leads to lower *task duration* for the mechanism without bandwidth control.

In Figure 5.9, experimental measurements are also compared against the theoretical lower-bound computed as described in Section 5.2. As the graphs show, the lower-bound values are sometimes higher than experimental results. This is because the value of lower-bound for *task duration time* is computed by dividing the theoretically computed *job completion time* lower-bound by the number of tasks, thus leading to an heuristic approximation and not to an absolute optimised lower-bound value. Figure 5.10A displays the actual lower-bound value for *job duration*, computed as described in Section 5.2.

The summarising results displayed in Figure 5.10 also highlight the relatively small benefits brought

Number GAs	F-value	p-value
1 GA	F(2,12)=10.366	p=0.002428
2 GAs	F(2,27)=33.28	p=5.173e-08
3 GAs	F(2,24)=17.569	p=2.855e-06
4 GAs	F(2,57)=24.575	p=2.012e-08
5 GAs	F(2,72)=14.128	p=6.667e-06

Table 5.4: Anova analysis for job duration varying GA in I/O-bound jobs (Figure 5.13A).

Number GAs	F-value	p-value
1 GA	F(3,116)=40.345	p=2.2e-16
2 GAs	F(3,116)=57.798	p=2.2e-16
3 GAs	F(3,116)=21.996	p=2.393e-11
4 GAs	F(3,116)=18.702	p=5.799e-10
5 GAs	F(3,116)=10.399	p=4.11e-06

Table 5.5: Anova analysis for task duration varying GA in I/O-bound jobs (Figure 5.13B).

by the more sophisticated P-BATS heuristic, as compared to the simpler BATS one, for both *job duration* and *task duration*. In fact, whenever multiple GEs simultaneously compete to access to a TE's shared resources, they are responsible for introducing an allocation unpredictability decreasing the benefits of introducing more sophistication. The GA can never be totally sure of the match between TE's available resources and real-time allocation, since other GAs could interfere among each other during the allocation process.

Finally, again in Figure 5.10, observe how the theoretical lower-bound seems to significantly outperform experimental results. In reality, the gap illustrates the major impact of radio communication interference, message collision and retransmission mechanisms, occurring under heavy communication load, in causing critical and prolonged delays that are not easy to model through simulations. In particular, since for the current experimental setting (i.e. offload of I/O-bound jobs) radio communication is expected to play a crucial role, the displayed experimental gap between theoretical lower-bound and measurements meets the expectations. The behaviours explained above are confirmed by the Anova results summarised in Tables 5.2 and 5.3.

Effect of varying the number of the GAs: The aim of this experiment is to show the nature of the speed-up obtained if the number of the GAs, simultaneously offloading I/O-bound jobs within the system, increases.

Each heterogeneous tasks of which the I/O-bound job is comprised is profiled according to the distribution of values listed in Table 5.1. The number of the GAs increases from 1 to 5. The number of the TEs is kept fixed at 6. In particular, 3 TEs are located in the area affected by Str, while the other 3 TEs are positioned in the area free from Str influence.

As illustrated in Figures 5.11, 5.12 and 5.13, considerations similar to those listed while investigating the effects of varying the number of the TEs can be derived for the metrics *task completion time*, *job duration* and *task duration*.

In particular, as shown in Figure 5.11, the increasing GA number reduces the performance gap between the two cases with and without bandwidth control for both heuristic mechanisms. This is as expected, since the available TE resources remain constant, despite the increasing GA number.

Let us now analyse Figures 5.11A. While in most of the situations, in fact, the BATS and P-BATS heuristics show comparable performance, in Figure 5.11A, and thus in the situation in which a single GA

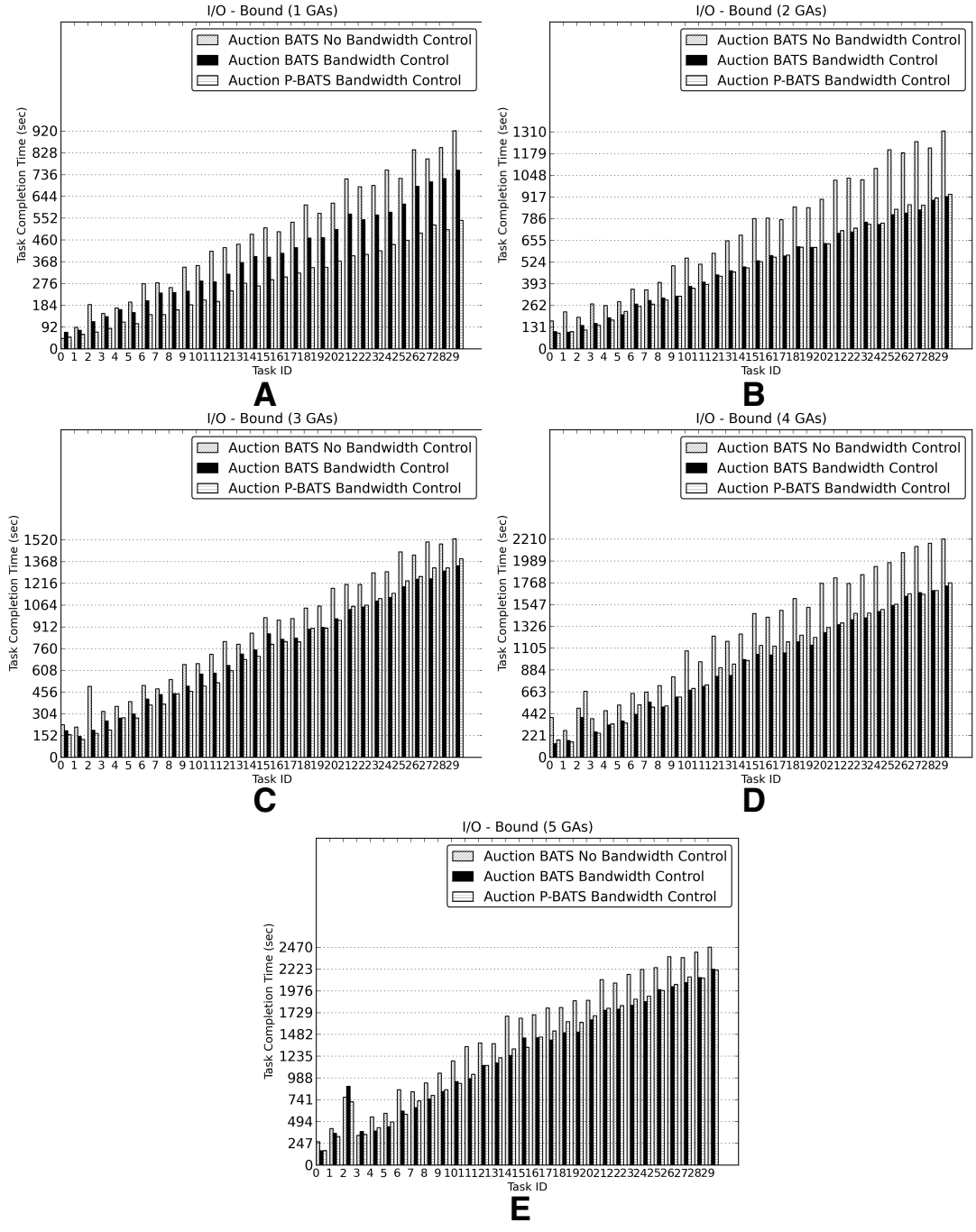


Figure 5.11: Effects of varying GA and $Task_{ID}$ on task completion time in I/O-bound jobs.

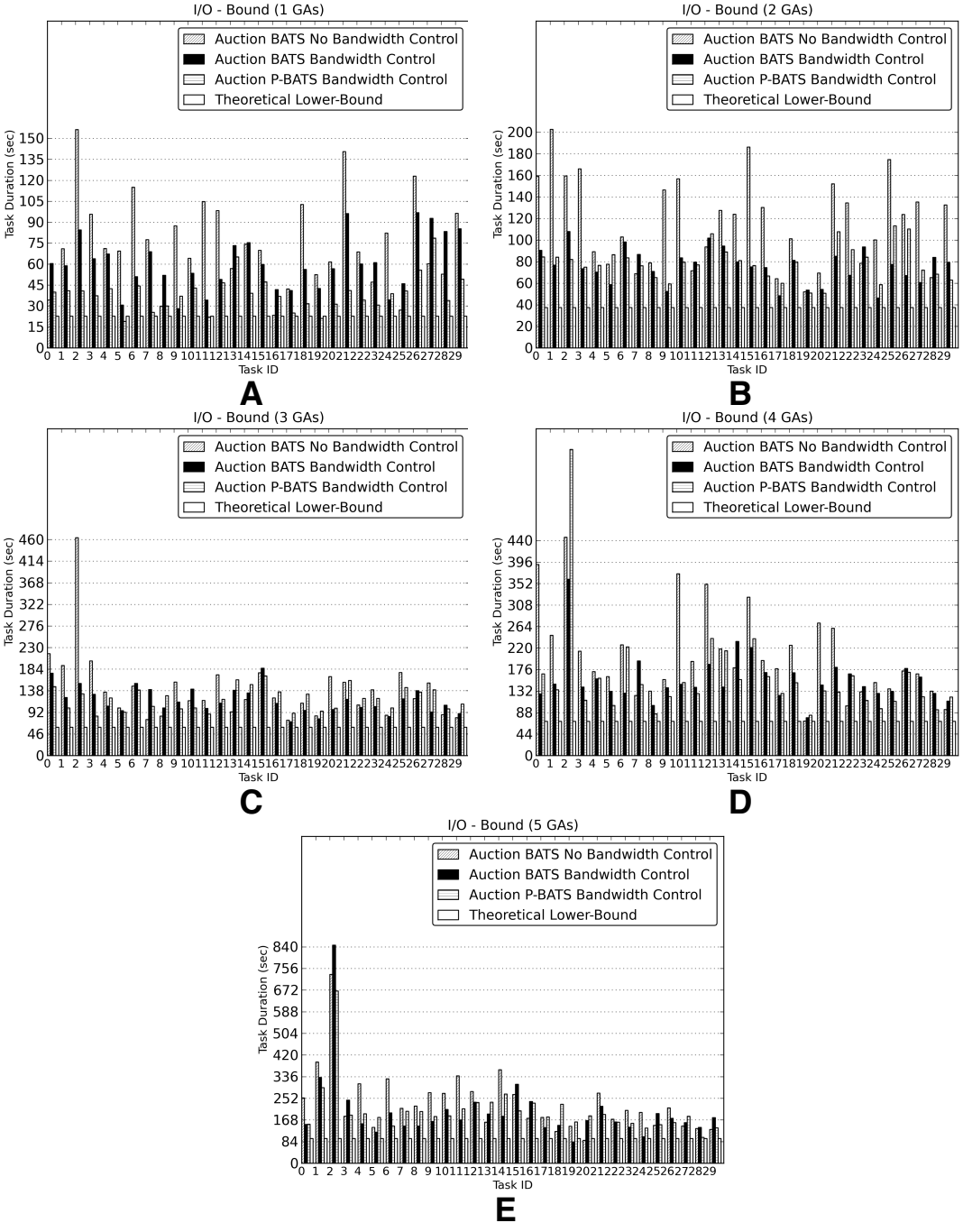


Figure 5.12: Effects of varying GA and $Task_{ID}$ on task duration in I/O-bound jobs.

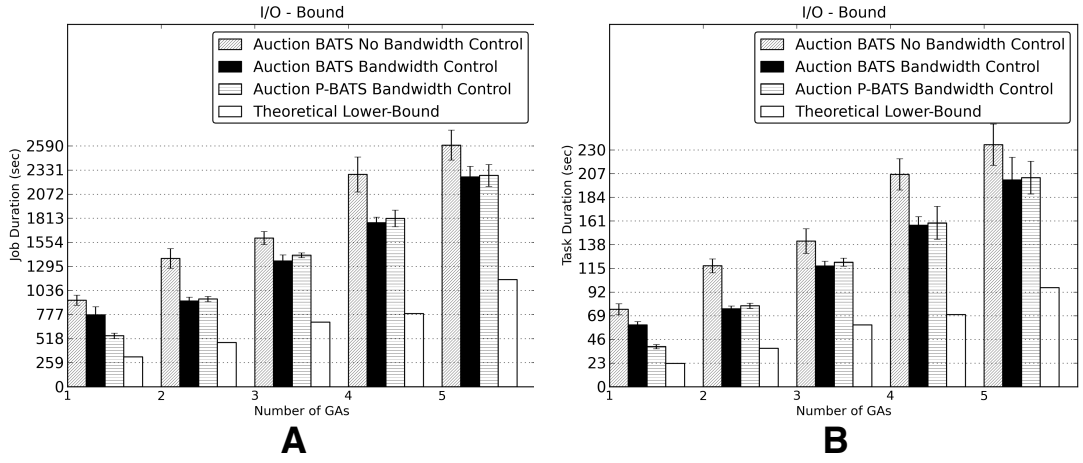


Figure 5.13: Effects of varying GA on job duration and task duration in I/O-bound jobs.

is responsible for load distributing jobs, P-BATS outperforms BATS. The reason is that whenever a single GA exclusively offloads tasks, it has a full and total control of TEs resources (resources not required to be shared among several GAs) and thus it profits by P-BATS sophistication since there is always a real-time match between advertised and actual TE resources. On the other hand, whenever multiple GEs simultaneously compete to access to TEs' shared resources, they are responsible for introducing an allocation unpredictability responsible for decreasing such benefits. In fact, the GA can never be totally sure of the match between TEs available resources and real-time allocation since other GAs could have, in the meantime, modified TEs status. This explains why improvements are not detected when a set of GAs simultaneously offload.

The anomalous behaviour of the first tasks in Figures 5.11D-E, and corresponding Figures 5.12D-E, is explained as above. Thus, whenever the GAs simultaneously start offloading, the TEs find themselves dealing with a sudden burst of requests. The least congested TEs quickly saturate their computational slots and some of the tasks have no choice but to be offloaded towards more congested TEs. This leads to a peak of the *task completion time* and *task duration* for the first tasks to be computed which then quickly normalises as the experiment progresses. Clearly, if the available TEs are sufficient to handle the initial burst of requests, the pattern is obfuscated (e.g. Figures 5.11A-B-C and Figures 5.12A-B-C). The behaviours explained above are confirmed by the Anova results summarised in Tables 5.4 and 5.5.

5.4.3 3^{rd} Experimental Setup and Results (CPU-Bound Jobs)

The aim of this third set of experiments is the investigation of the impact of network conditions in dealing with the distribution of CPU-bound applications through DWAG collaborations. The experimental setup is as described in Table 5.6, thus we have up to 5 GAs and 10 TEs. The former are pre-loaded with CPU-bound jobs, each of which split into 30 tasks. Each task is formed by 30 repetitions of the actions (i) burst of offload messages, (ii) computation cycles, and (iii) burst of upload messages.

As mentioned in Section 5.4.2, the values for I/O bursts and computation cycles are randomly chosen within the Gaussian (communication) and Poissonian (computation) distributions. The values,

CPU-bound job setup	
Number GAs	5
Number TEs	10
Number Tasks	30
Repetitions	30
Gaussian σ	1
Gaussian μ	2
Poissonian λ	10

Table 5.6: Experimental setup for CPU-bound job experimentation.

randomly picked within the distributions, remain constant over the task execution for each repeated cycle, but they instead change for every task. Hence, once a set of values for all tasks has been selected, the same sequence is used to allow experimental repeatability. The values adopted within the distributions are as follows: $\sigma = 1$ and $\mu = 2$ for the Gaussian distribution, $\lambda = 10$ for the Poissonian distribution. The values of the weights within P-BATS have been set to $weight_{Band} = 1$ and $weight_{CPU} = 2$ for CPU-bound jobs and they have been computed adopting the same procedure as that used for BATS, but applied to P-BATS. Again, experimental parameters have been set in this way according to range variations of the experiments in Chapter 4.

Effect of varying the number of the TEs: The aim of this experiment is to show the nature of the speed-up obtained if the number of the TEs available for computation in the system increases.

Again, we vary the number of the TEs from 4 to 10 progressively adding 2 TEs at the time and locating each one of them in the two diversified areas, affected or not by Str's generated network congestion. In this experiment, 4 GAs simultaneously offload CPU-bound jobs each of which composed of heterogeneous tasks profiled according to the distribution values detailed in Table 5.6. The number of the GAs is kept fixed within the experiment.

A very interesting behaviour arises though by observing Figures 5.14 and 5.16. In fact, as can be seen, in terms of *task completion time* (Figure 5.14) and *job duration* (Figure 5.16A), the performance gap between bandwidth and no bandwidth control cases is not as pronounced as for the *task duration*. As illustrated in Figures 5.15 and 5.16B, considerable improvements are seen in the *task duration* by applying BATS and P-BATS heuristics with bandwidth control against BATS without bandwidth control. In particular, both BATS and P-BATS with bandwidth control approximatively halve task duration values. The reason for such results is that, whenever CPU-bound tasks must be computed, they are completed almost instantaneously with a negligible number of message exchanges thus, as expected, a limited performance gap between bandwidth and no bandwidth control is detected in *task completion time* and *job duration*. Thus, despite the benefits brought by accounting for network information, the pronounced *task duration* gap is likely to be linked to the overhead introduced by the distribution algorithm. In fact, before distributing, the infrastructure needs to perform a negotiation phase to collect network details and take an informed decision based on them. On CPU-bound jobs though such network information has a

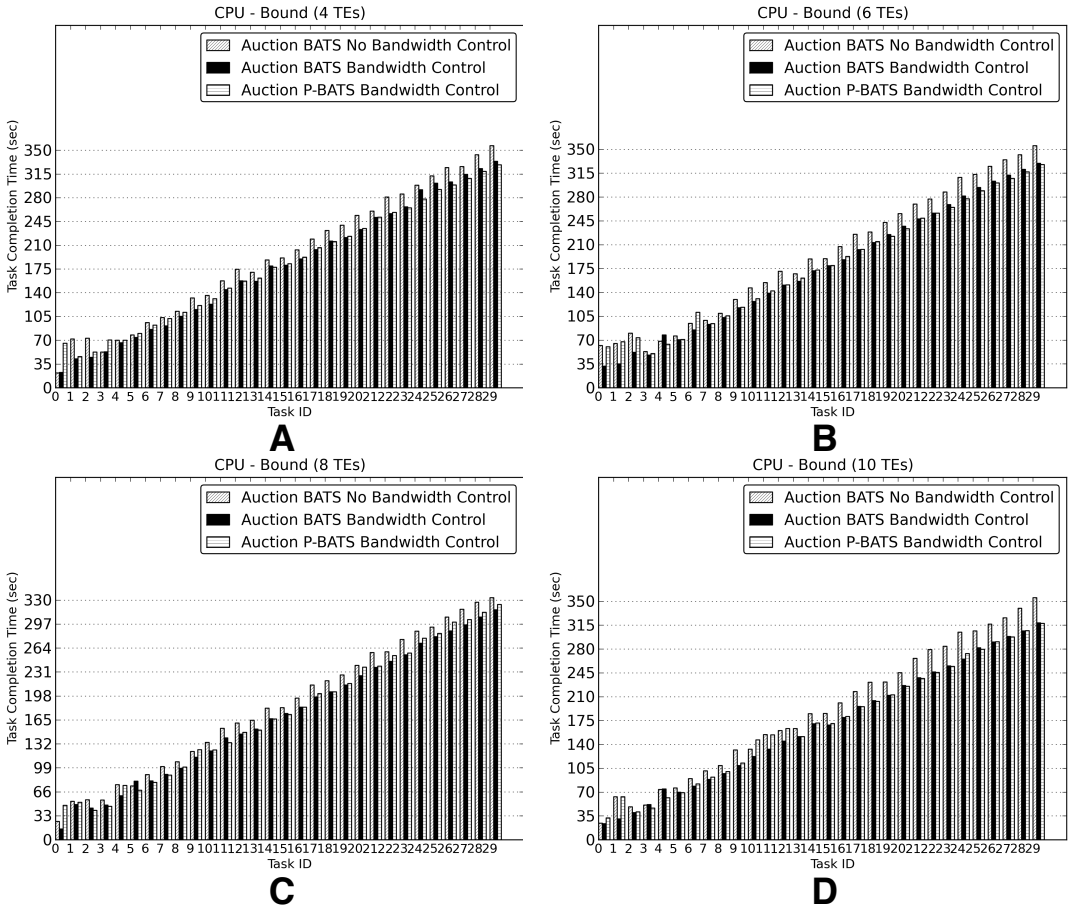


Figure 5.14: Effects of varying TE and $Task_{ID}$ on task completion time in CPU-bound jobs.

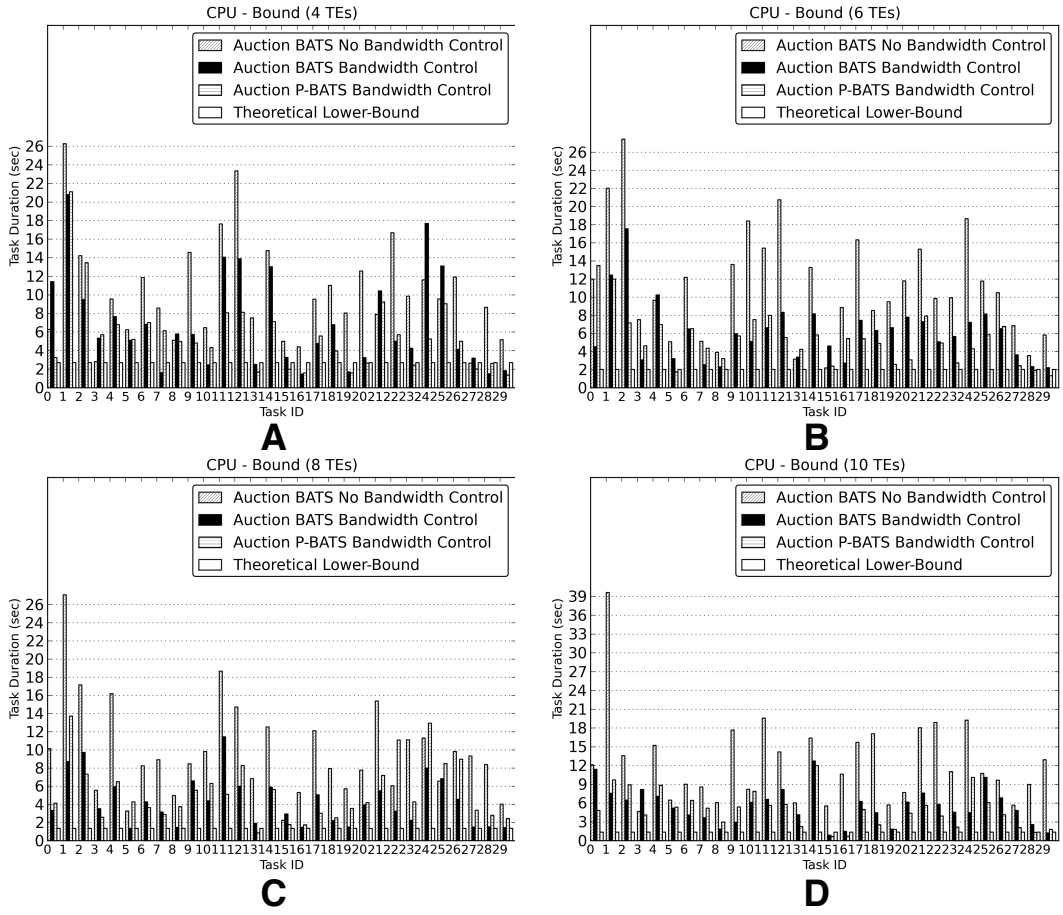


Figure 5.15: Effects of varying TE and $Task_{ID}$ on task duration in CPU-bound jobs.

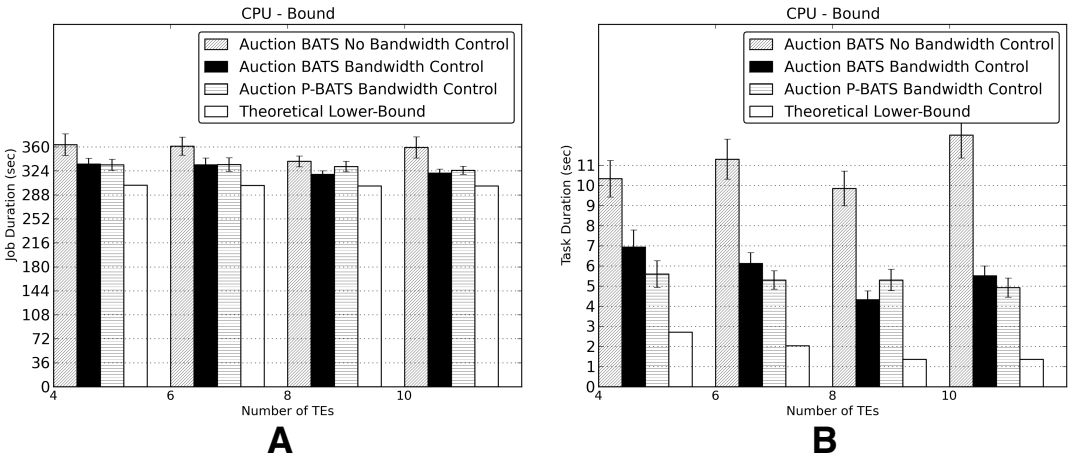


Figure 5.16: Effects of varying TE on job duration and task duration in CPU-bound jobs.

Number TEs	F-value	p-value
4 TEs	F(2,57)=10.499	p=0.0001312
6 TEs	F(2,57)=11.028	p=8.936e-05
8 TEs	F(2,57)=8.7777	p=0.0004751
10 TEs	F(2,69)=24.223	p=1.074e-08

Table 5.7: Anova analysis for job duration varying TE in CPU-bound jobs (Figure 5.16A).

Number TEs	F-value	p-value
4 TEs	F(3,116)=16.235	p=7.09e-09
6 TEs	F(3,116)=31.847	p=4.31e-15
8 TEs	F(3,116)=32.614	p=2.316e-15
10 TEs	F(3,116)=39.733	p=2.2e-16

Table 5.8: Anova analysis for task duration varying TE in CPU-bound jobs (Figure 5.16B).

limited impact, thus the biggest price to be paid is in fetching and negotiating such information, through the load distribution algorithms.

In Figure 5.15, experimental measurements are also compared against the theoretical lower-bound computed as described in Section 5.2. Once again, as illustrated by the graphs, the lower-bound values happen here to be sometimes higher than experimental results. This is because the value of lower-bound for *task duration time* is computed by dividing the theoretically computed *job completion time* lower-bound for the number of tasks, thus leading to an heuristic approximation and not to an absolute optimised lower-bound value. Figure 5.16A displays the lower-bound for *job duration*, computed as described in Section 5.2. Let us notice how, in terms of *job duration* (Figure 5.16A), the theoretical lower-bound is comparable to the actual measured results. This is because only CPU-bound jobs are distributed and thus the effect of network traffic and congestion are contained. The behaviours explained above are confirmed by the Anova results summarised in Tables 5.7 and 5.8.

Effect of varying the number of the GAs: The aim of this experiment is to show the nature of the speed-up obtained if the number of the GAs, simultaneously offloading CPU-bound jobs within the system, increases.

Each heterogeneous tasks of which the CPU-bound job is comprised is profiled according to the distribution values listed in Table 5.6. The number of the GAs increases from 1 to 5. The number of the TEs is fixed at 6. In particular, 3 TEs are located in the area affected by Str, while the other 3 TEs are positioned in the area free from Str influence.

As illustrated in Figures 5.17, 5.18 and 5.19, considerations similar to those listed while investigating the effects of varying the number of the TEs can be driven for the metrics *task completion time*, *job duration* and *task duration*.

Both BATS and P-BATS heuristics with bandwidth control outperform the case without bandwidth control. The performance gap for *task completion time* increases by increasing the number of the GAs, since additional load must be distributed on a constant number of TE resources. This trend, partially obfuscated in Figure 5.17A-B-C-D, is clearly displayed in Figure 5.17E. Again, the pronounced gap in *task duration* is linked to the additional overhead introduced by the negotiation phase within the load distribution algorithm. In fact, since, within CPU-bound jobs, communication is negligible with respect to computation and the latter is quickly performed, then the overhead introduced by the load sharing algorithms to fetch network information plays a role.

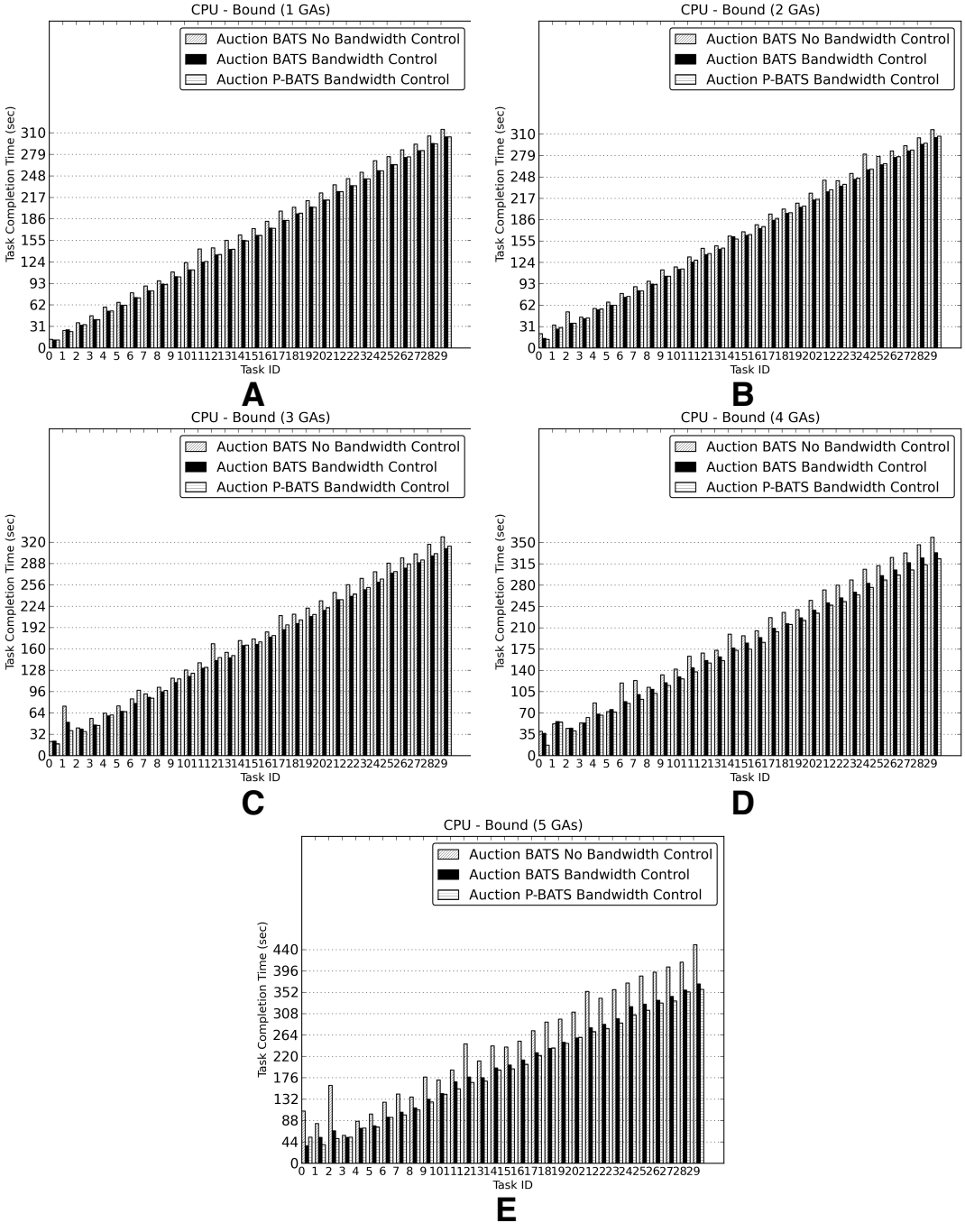


Figure 5.17: Effects of varying GA and $Task_{ID}$ on task completion time in CPU-bound jobs.

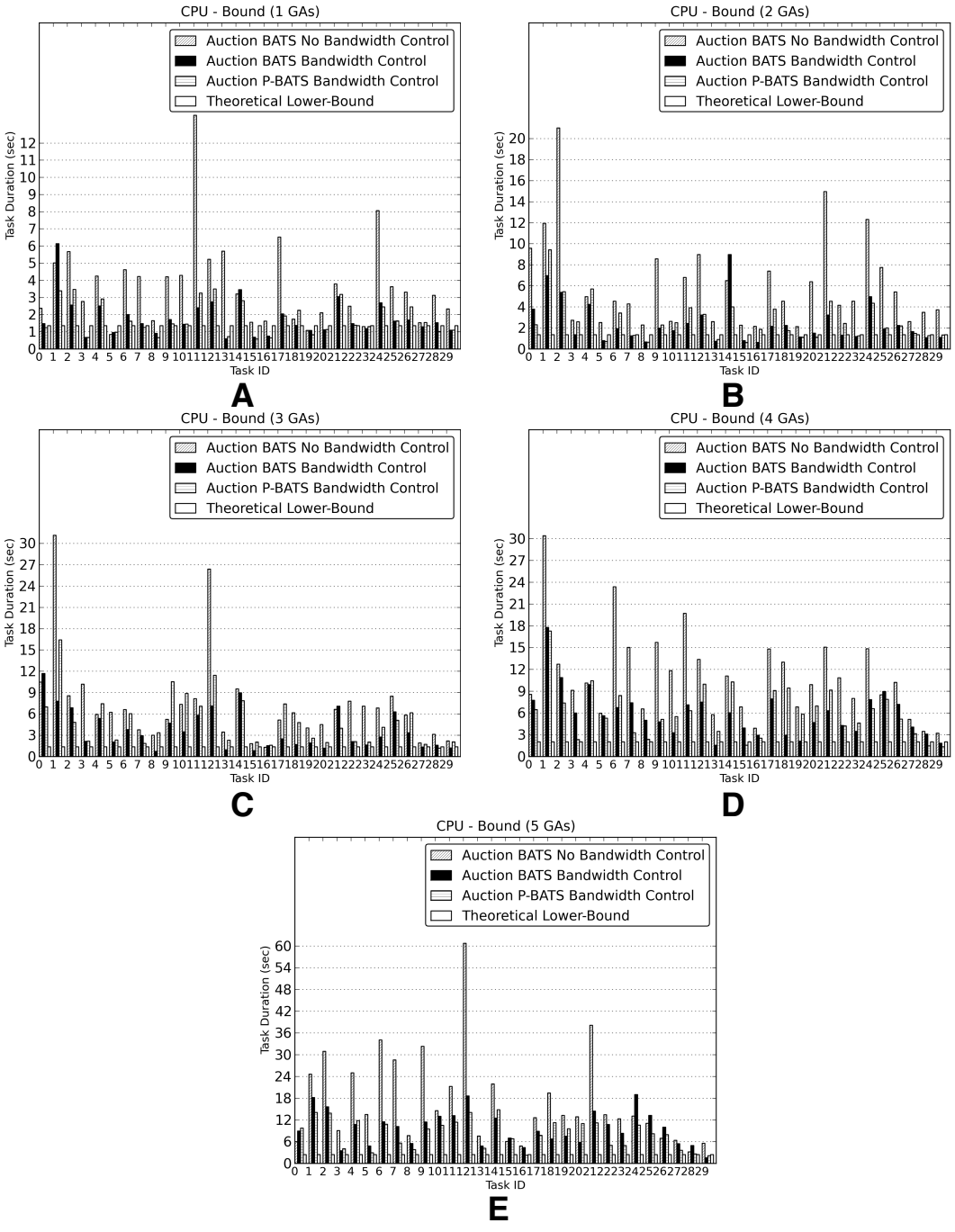


Figure 5.18: Effects of varying GA and $Task_{ID}$ on task duration in CPU-bound jobs.

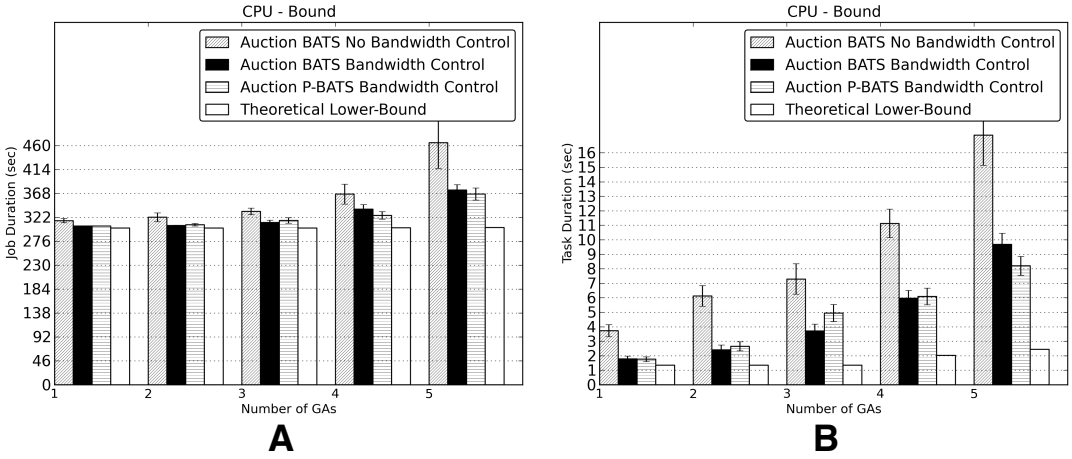


Figure 5.19: Effects of varying GA on job duration and task duration in CPU-bound jobs.

Number GAs	F-value	p-value
1 GA	F(2,12)=6.1958	p=0.01418
2 GAs	F(2,27)=6.6468	p=0.004495
3 GAs	F(2,42)=15.407	p=9.585e-06
4 GAs	F(2,57)=12.945	p=2.317e-05
5 GAs	F(2,72)=20.309	p=1.014e-07

Table 5.9: Anova analysis for job duration varying GA in CPU-bound jobs (Figure 5.19A).

Number GAs	F-value	p-value
1 GA	F(3,116)=15.576	p=1.408e-08
2 GAs	F(3,116)=19.252	p=3.364e-10
3 GAs	F(3,116)=11.988	p=6.812e-07
4 GAs	F(3,116)=28.32	p=8.186e-14
5 GAs	F(3,116)=22.321	p=1.763e-11

Table 5.10: Anova analysis for task duration varying GA in CPU-bound jobs (Figure 5.19B).

In Figure 5.18, the lower-bound values are sometimes higher than experimental results. This is as mentioned above. Figure 5.19A displays the lower-bound for *job duration*, computed as described in Section 5.2. Again, in terms of *job duration* (Figure 5.19A), the theoretical lower-bound is comparable to the actual measured results, since only CPU-bound jobs are distributed and thus the effect of network traffic and congestion are contained. The behaviours explained above are confirmed by the Anova results summarised in Tables 5.9 and 5.10.

5.4.4 4^{th} Experimental Setup and Results (Balanced Jobs)

The aim of this fourth set of experiments is the investigation of the impact of network conditions in dealing with the distribution of balanced kinds of applications through DWAG collaborations. The experimental setup is as described in Table 5.11, thus we have up to 5 GAs and 10 TEs. The former are pre-loaded with balanced jobs each of which split into 30 tasks. Each task is formed by 30 repetitions of the actions (i) burst of offload messages, (ii) computation cycles, and (iii) burst of upload messages.

As aforementioned, the values for I/O bursts and computation cycles are randomly chosen within the Gaussian (communication) and Poissonian (computation) distributions. The values, randomly picked within the distributions, remain constant over the task execution for each repeated cycle, but they instead change for every task. Hence, once a set of values for all tasks has been selected, the same sequence

Balanced job setup	
Number GAs	5
Number TEs	10
Number Tasks	30
Repetitions	30
Gaussian σ	5
Gaussian μ	30
Poissonian λ	10

Table 5.11: Experimental setup for balanced job experimentation.

is used to allow experimental repeatability. The values adopted within the distributions are as follows: $\sigma = 5$ and $\mu = 30$ for the Gaussian distribution, $\lambda = 10$ for the Poissonian distribution. The values of the weights within P-BATS have been set to $weight_{Band} = 1$ and $weight_{CPU} = 450$ for balanced jobs and they have been computed adopting the same procedure as that used for BATS, but applied to P-BATS. Again, experimental parameters have been set in this way according to range variations of the experiments in Chapter 4.

Effect of varying the number of the TEs: The aim of this experiment is to show the nature of the speed-up obtained if the number of the TEs available for computation in the system increases.

Again, we vary the number of the TEs from 4 to 10 progressively adding 2 TEs at the time and locating each one of them in the two areas of influence generated by the Str node. In this experiment, 4 GAs simultaneously offload balanced jobs each of which composed of heterogeneous tasks profiled according to the distribution values detailed in Table 5.11. The number of the GAs is kept fixed within the experiment.

In these experiments, we distribute in the environment balanced jobs. As expected, experimental results gracefully combine both behaviours described for I/O-bound (Section 5.4.2) and CPU-bound (Section 5.4.3) jobs. As illustrated in Figure 5.20, in terms of *task completion time* both BATS and P-BATS heuristics with congestion control outperform the case in which only TEs computational requirements are accounted for. Increasing the number of the TEs reduces the performance gap between the two cases with and without bandwidth control for both heuristic mechanisms, since the computational load to be distributed remains constant.

As illustrated in Figure 5.21, considerations similar to those regarding *task completion time* can be propagated to *task duration*. Sometimes the *task duration* for heuristics with bandwidth control display increased values with respect to the case where bandwidth control is disregarded. This happens because the mechanism distributing load exclusively accounting for TEs computational capabilities does not exclusively offload tasks onto congested TEs. In fact, it might happen that a node more capable from a computational perspective is also located within a uncongested area. Thus, the overlap of conditions leads to lower *task duration* for the mechanism without bandwidth control.

The considerations concerning lower-bound values in Figures 5.21 and 5.22 are described as above.

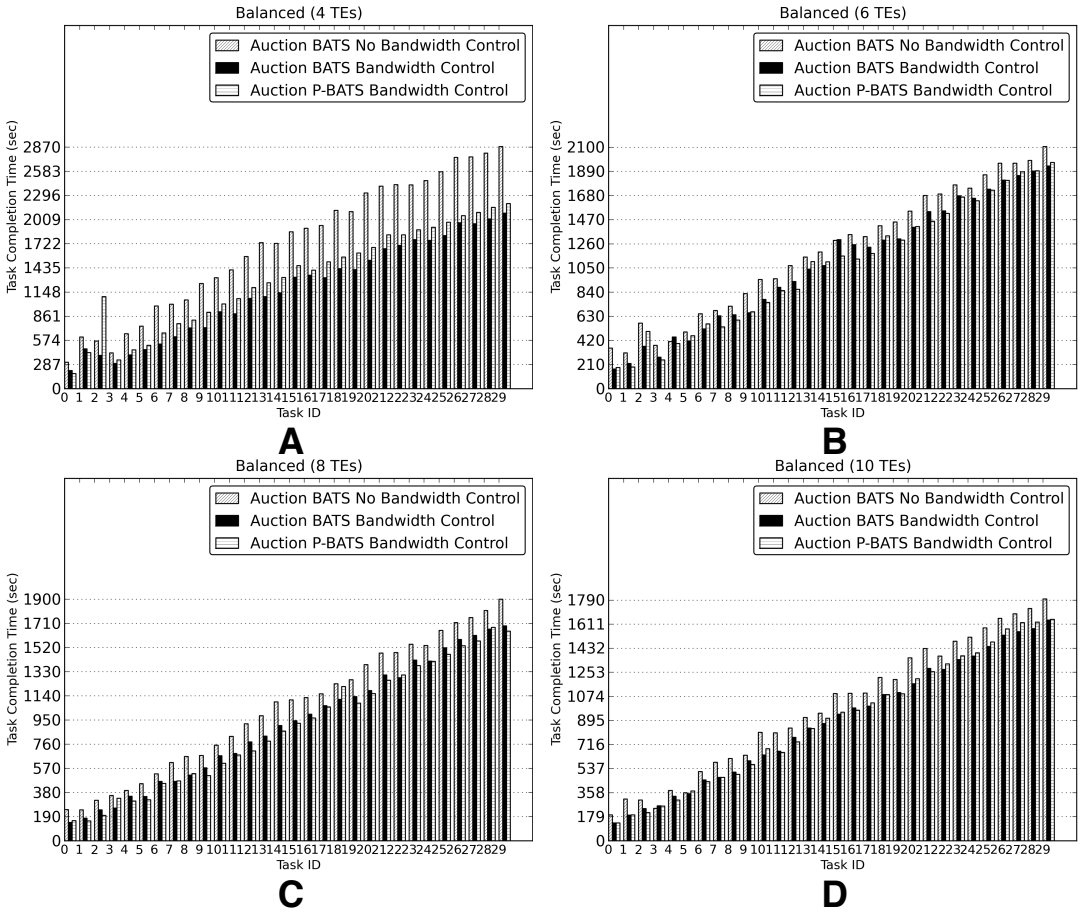


Figure 5.20: Effects of varying TE and $Task_{ID}$ on task completion time in balanced jobs.

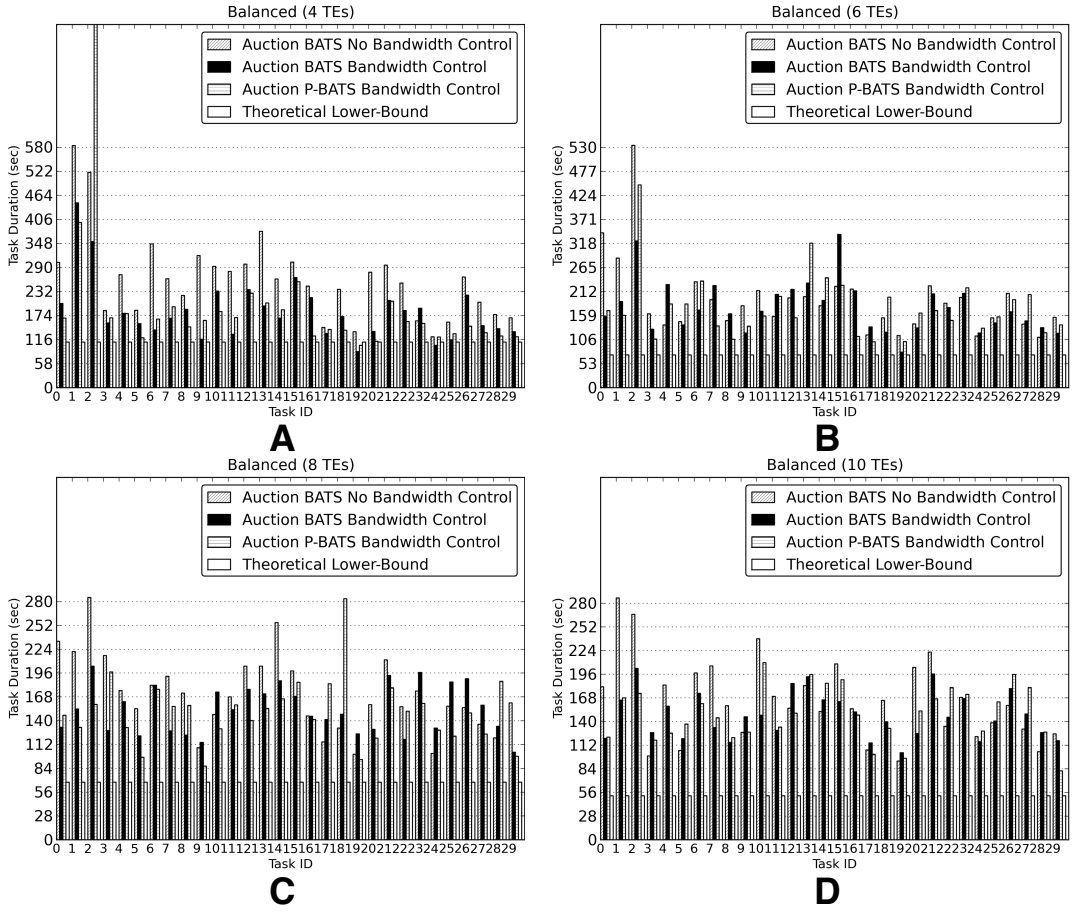


Figure 5.21: Effects of varying TE and $Task_{ID}$ on task duration in balanced jobs.

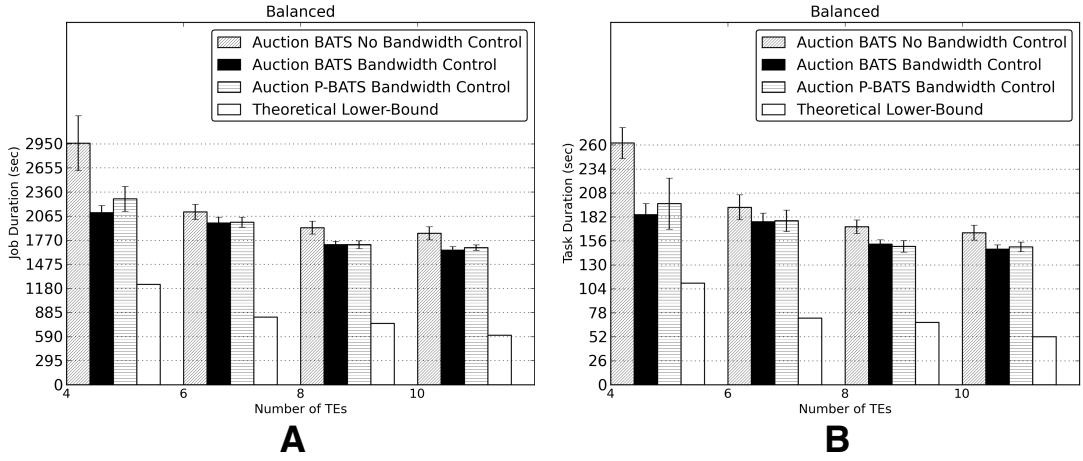


Figure 5.22: Effects of varying TE on job duration and task duration in balanced jobs.

Number TEs	F-value	p-value
4 TEs	F(2,69)=24.894	p=7.256e-09
6 TEs	F(2,81)=9.3359	p=0.0002246
8 TEs	F(2,57)=19.406	p=3.733e-07
10 TEs	F(2,57)=19.225	p=4.157e-07

Table 5.12: Anova analysis for job duration varying TE in balanced jobs (Figure 5.22A).

Number TEs	F-value	p-value
4 TEs	F(3,116)=10.490	p=3.699e-06
6 TEs	F(3,116)=24.484	p=2.401e-12
8 TEs	F(3,116)=59.743	p=2.2e-16
10 TEs	F(3,116)=76.938	p=2.2e-16

Table 5.13: Anova analysis for task duration varying TE in balanced jobs (Figure 5.22B).

Finally, as illustrated in Figure 5.22, the theoretical lower-bound seems to significantly outperform experimental results. In reality, the gap illustrates the major impact of radio communication interference, message collision and retransmission mechanisms, occurring under heavy communication load, in causing inevitably critical and prolonged delays with difficulty modelled through simulations. In particular, since for the current experimental setting (i.e. offload of balanced jobs) radio communication is expected to play a crucial role, the displayed experimental gap between theoretical lower-bound and measurements meets the expectations. The behaviours explained above are confirmed by the Anova results summarised in Tables 5.12 and 5.13.

Effect of varying the number of the GAs: The aim of this experiment is to show the nature of the speed-up obtained if the number of the GAs, simultaneously offloading I/O-bound jobs within the system, increases.

Each heterogeneous tasks of which the balanced job is comprised is profiled according to the distribution values listed in Table 5.11. The number of the GAs increases from 1 to 5. The number of the TEs is kept fixed to 6. In particular, 3 TEs are located in the area affected by Str, while the other 3 TEs are positioned in the area free from Str influence.

Again, for balanced jobs results mediate both I/O-bound (Section 5.4.2) and CPU-bound (Section 5.4.3) behaviours. As illustrated in Figure 5.23, increasing the number of the GAs reduces the performance gap between the two cases with and without bandwidth control for both heuristic mechanisms. This is as expected, since the available TE resources remain constant, despite the increasing GA number.

Let us now analyse the behaviour displayed in Figures 5.23A. While in most of the situations the P-BATS and BATS heuristics show comparable performance, in Figure 5.11A, and thus in the situation in which a single GA is responsible for load distributing jobs, BATS outperforms P-BATS. This happens because the more sophisticated computation, that must be performed by the P-BATS heuristic, increases the computational overhead that must be handled by devices leading nodes towards consuming more internal resources, spending more time for the computation itself, thus leading the GAs to make decisions based on inevitable out-of-date information. This pattern, so clearly displayed when a single GA is load distributing (Figure 5.23A), is instead obfuscated when multiple GAs simultaneously offload computation (Figure 5.23B-C-D-E). The behaviours explained above are confirmed by the Anova results summarised in Tables 5.14 and 5.15.

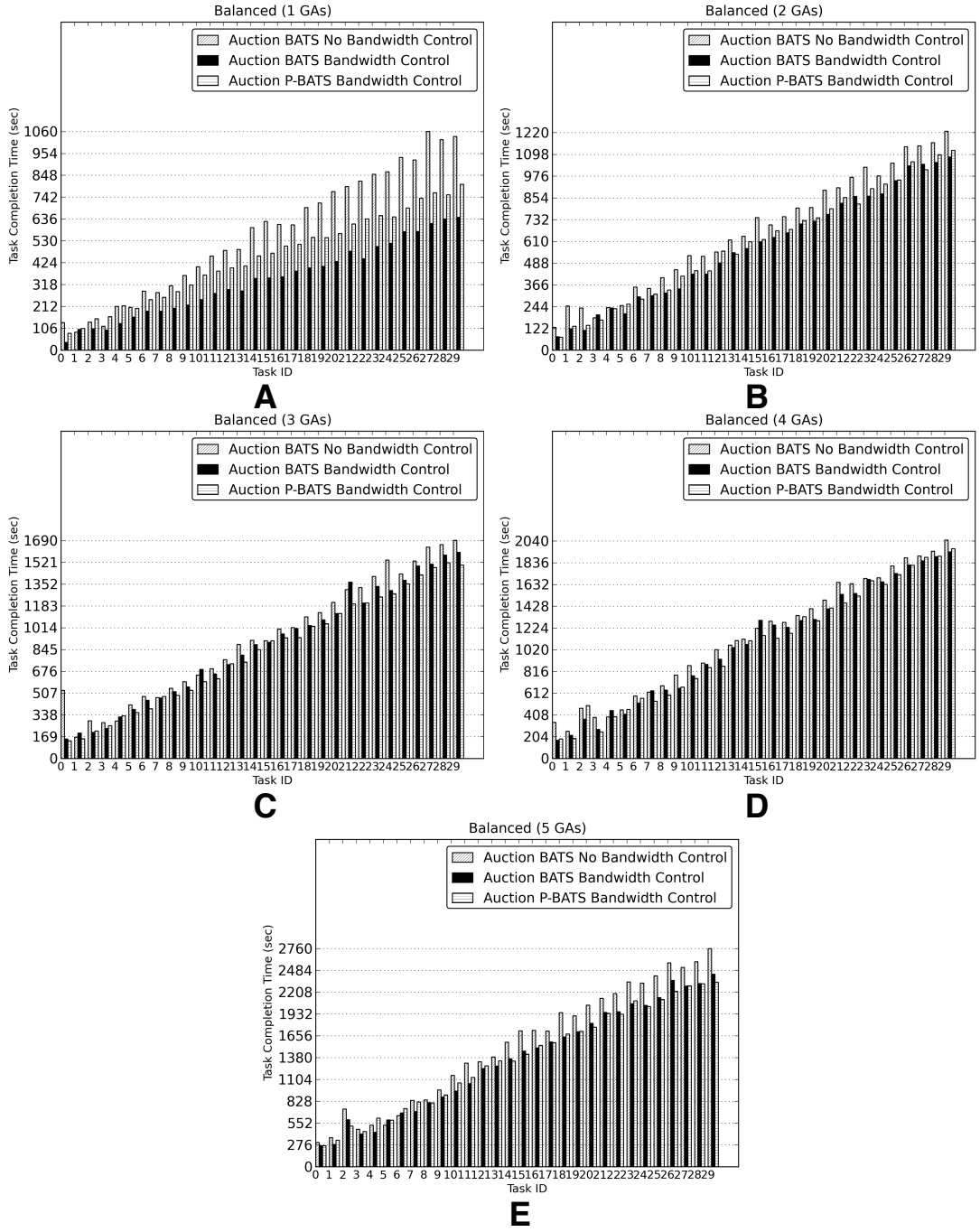


Figure 5.23: Effects of varying GA and $Task_{ID}$ on task completion time in balanced jobs.

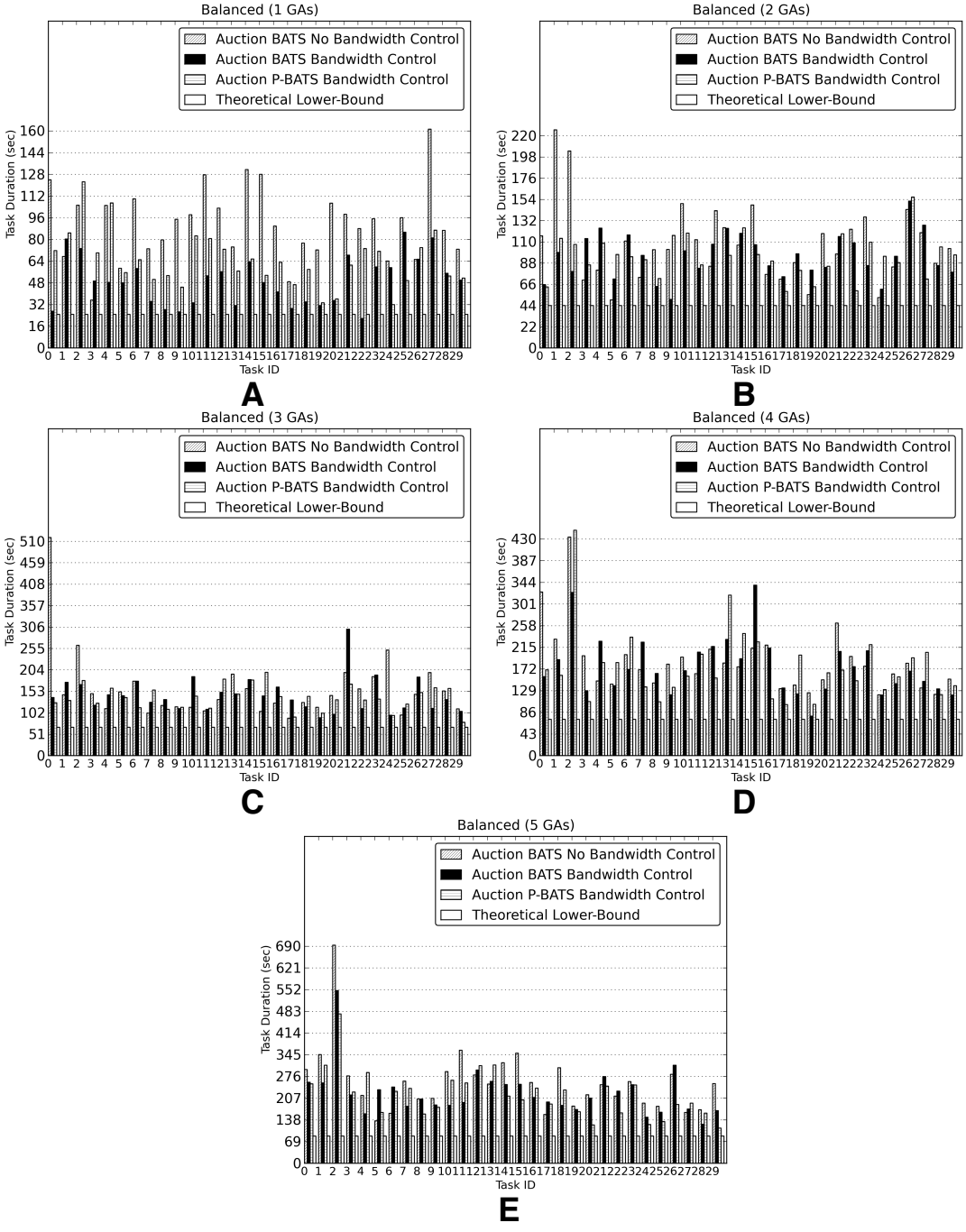


Figure 5.24: Effects of varying GA and $Task_{ID}$ on task duration in balanced jobs.

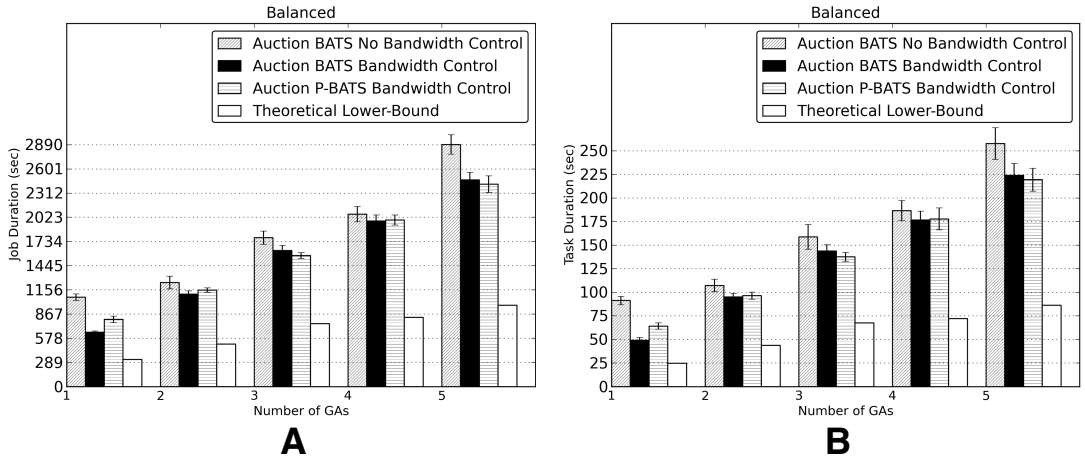


Figure 5.25: Effects of varying GA on job duration and task duration in balanced jobs.

Number GAs	F-value	p-value
1 GA	F(2,12)=41.984	p=3.822e-06
2 GAs	F(2,12)=11.147	p=0.001835
3 GAs	F(2,60)=13.538	p=1.404e-05
4 GAs	F(2,12)=14.037	p=0.000721
5 GAs	F(2,72)=38.227	p=4.861e-12

Table 5.14: Anova analysis for job duration varying GA in balanced jobs (Figure 5.25A).

Number GAs	F-value	p-value
1 GA	F(3,116)=63.991	p=2.2e-16
2 GAs	F(3,116)=35.903	p=2.2e-16
3 GAs	F(3,116)=21.774	p=2.951e-11
4 GAs	F(3,116)=28.3	p=8.33e-14
5 GAs	F(3,116)=31.199	p=7.325e-15

Table 5.15: Anova analysis for task duration varying GA in balanced jobs (Figure 5.25B).

Jobs	F-value	p-value
I/O-bound	F(2,12)=14.370	p=0.000653
CPU-bound	F(2,12)=4.7286	p=0.03059
Balanced	F(2,12)=12.358	p=0.001219

Table 5.16: Anova analysis for job duration varying job types (Figure 5.27A).

Jobs	F-value	p-value
I/O-bound	F(2,87)=6.9642	p=0.001565
CPU-bound	F(2,87)=0.0186	p=0.9815
Balanced	F(2,87)=3.8923	p=0.02404

Table 5.17: Anova analysis for task duration varying job types (Figure 5.27B).

5.4.5 5^{th} Experimental Setup and Results (Mixed Jobs)

The aim of this fifth set of experiments is the investigation of the impact of network conditions in dealing with the simultaneous distribution of heterogeneous job mixtures through DWAG collaborations. In particular, in the first configuration we have 3 GAs, each of which offloads one of the aforementioned jobs: I/O-bound, CPU-bound and balanced. In the second configuration, we have 6 GAs simultaneously offloading a job mixture, each two offloading tasks belonging to the same job category. The number of the TEs is fixed at 6. Jobs and tasks were configured according to the previously described I/O-bound (Section 5.4.2), CPU-bound (Section 5.4.3) and balanced (Section 5.4.4) settings. Each job was split into 30 tasks and each task is formed by 30 repetitions of the actions (i) burst of offload messages, (ii) computation cycles, and (iii) burst of upload messages.

Let us now summarise the main observations deriving from the experiment. Firstly, let us notice how the increase from 3 to 6 GAs simultaneously offloading tasks leads to a general performance decrease, in terms of *task completion time* (Figure 5.26A-C-E and 5.28A-C-E), *job duration* (Figure 5.27A and 5.29A) and *task duration* (Figure 5.27B and 5.29B). This occurs since a doubled load (from 3 to 6 GAs) must be distributed on a constant number of resources (6 TEs). However, as illustrated in Figures 5.26 and 5.28, for each job type considerable performance improvements are detected whenever heuristics accounting for bandwidth control are adopted against those without bandwidth control. This implies that a better allocation of the available resources, from a computational and a communication point of view, considerably speeds up the overall job computation. Moreover, it emphasises the importance of accounting for network conditions in deciding towards which part of the network the load should be distributed.

Secondly, as illustrated in Figures 5.27 and 5.29, it is possible to observe how the fine granularity brought by the more sophisticated P-BATS heuristic does not practically outperform for each job type the BATS one. In particular, in the case where 6 GAs are simultaneously load distributing tasks, it can be observed that the additional overhead introduced by the more refined P-BATS heuristic leads the system to consume more internal resources, spending more time for computation, thus leading to decisions based on out-of-date information.

Thirdly, as illustrated in Figures 5.27A and 5.29A, observe that the performance gaps between the cases with and without bandwidth control again reflect job patterns, each described within the aforementioned experiments. As expected, in the case with 6 GAs, the effect of performing considerable offloads (while keeping constant the number of TEs) combined with the injected traffic leads to a condition of

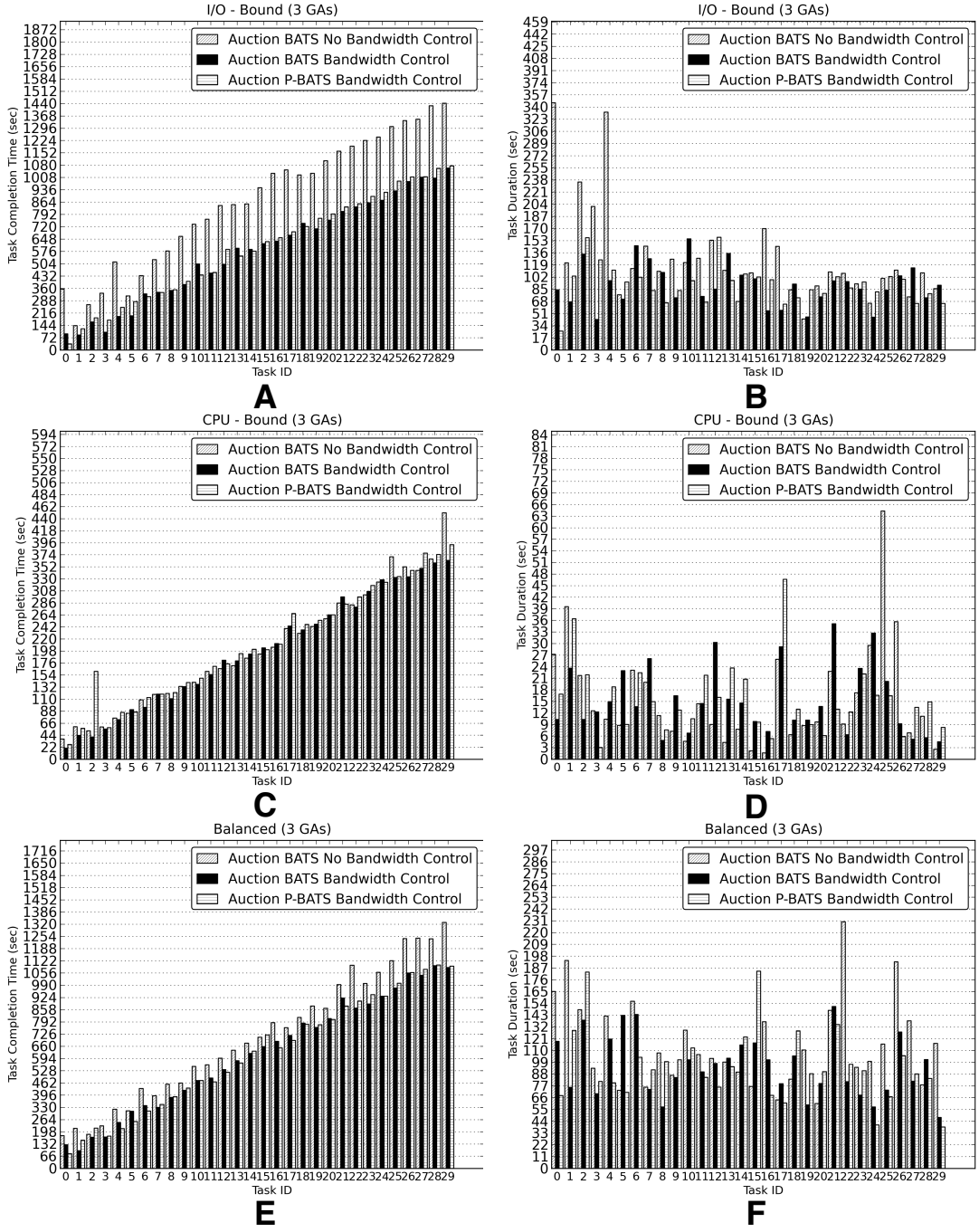


Figure 5.26: Effects of mixed job types on task completion time and task duration (3GAs).

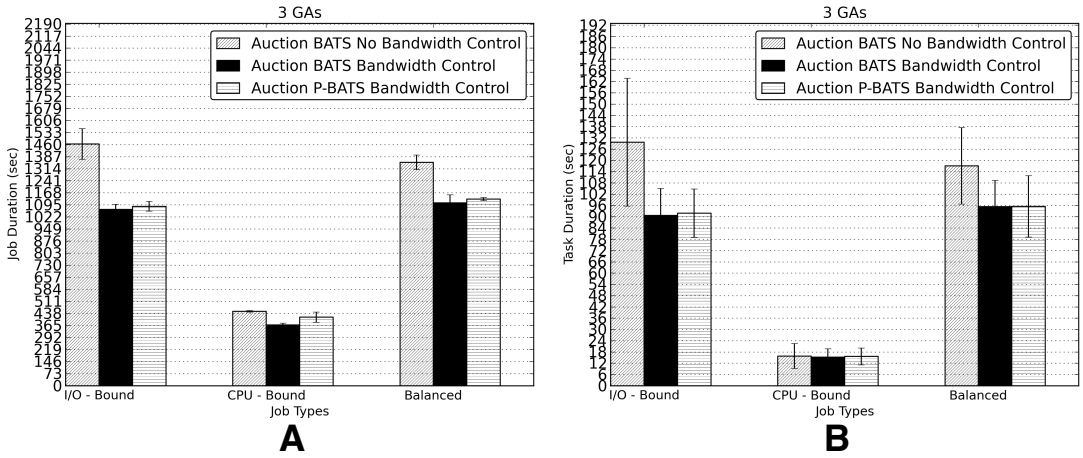


Figure 5.27: Effects of mixed job types on job duration and task duration (3GAs).

Job Type	F-value	p-value
I/O-bound	F(2,27)=10.420	p=0.0004427
CPU-bound	F(2,27)=2.4102	p=0.1089
Balanced	F(2,27)=3.6963	p=0.03812

Table 5.18: Anova analysis for job duration varying job types (Figure 5.29A).

Job Type	F-value	p-value
I/O-bound	F(2,87)=1.9501	p=0.1484
CPU-bound	F(2,87)=3.1981	p=0.04568
Balanced	F(2,87)=0.542	p=0.5835

Table 5.19: Anova analysis for task duration varying job types (Figure 5.29B).

generalised network saturation. Therefore, no real benefit is detected by applying any of the devised heuristics.

The behaviours explained above are confirmed by the Anova results summarised in Tables 5.16 and 5.17 for the case with 3 GAs and in Tables 5.18 and 5.19 for the case with 6 GAs.

Finally, let us now review some of the assumptions presented in Section 1.4 with the aim of speculating on system behaviour at the presence of constraint relaxation. If we did not assume constant radio connectivity over the timescale of each individual task offload, the overall *job duration* might increase due to the presence of retransmission mechanisms or the need to choose other nodes towards which to distribute computation. In this case it might, however, be convenient to expand the presented algorithms with additional control mechanisms so that, whenever nodes are moving sufficiently fast that radio connections are likely to break before offloaded tasks reach completion, it might still be possible to exploit multi-hop radio connectivity among them to disseminate the computed information without repeating the offload procedure. If we adopted heterogeneous nodes, we might contain the need for communication since nodes might be able to perform all the necessary computations. However, some applications are intrinsically distributed, therefore in such cases we do not expect gains in terms of *task completion time*, *job duration* and *task duration* by introducing heterogeneous nodes. If we introduced task prioritisation, we might assist in the reduction of execution delay or locking for some tasks, since those that are high priority might preempt their execution. Additional mechanisms would need to be introduced to allow blocked tasks to be executed on different nodes whenever a sequence of high priority would

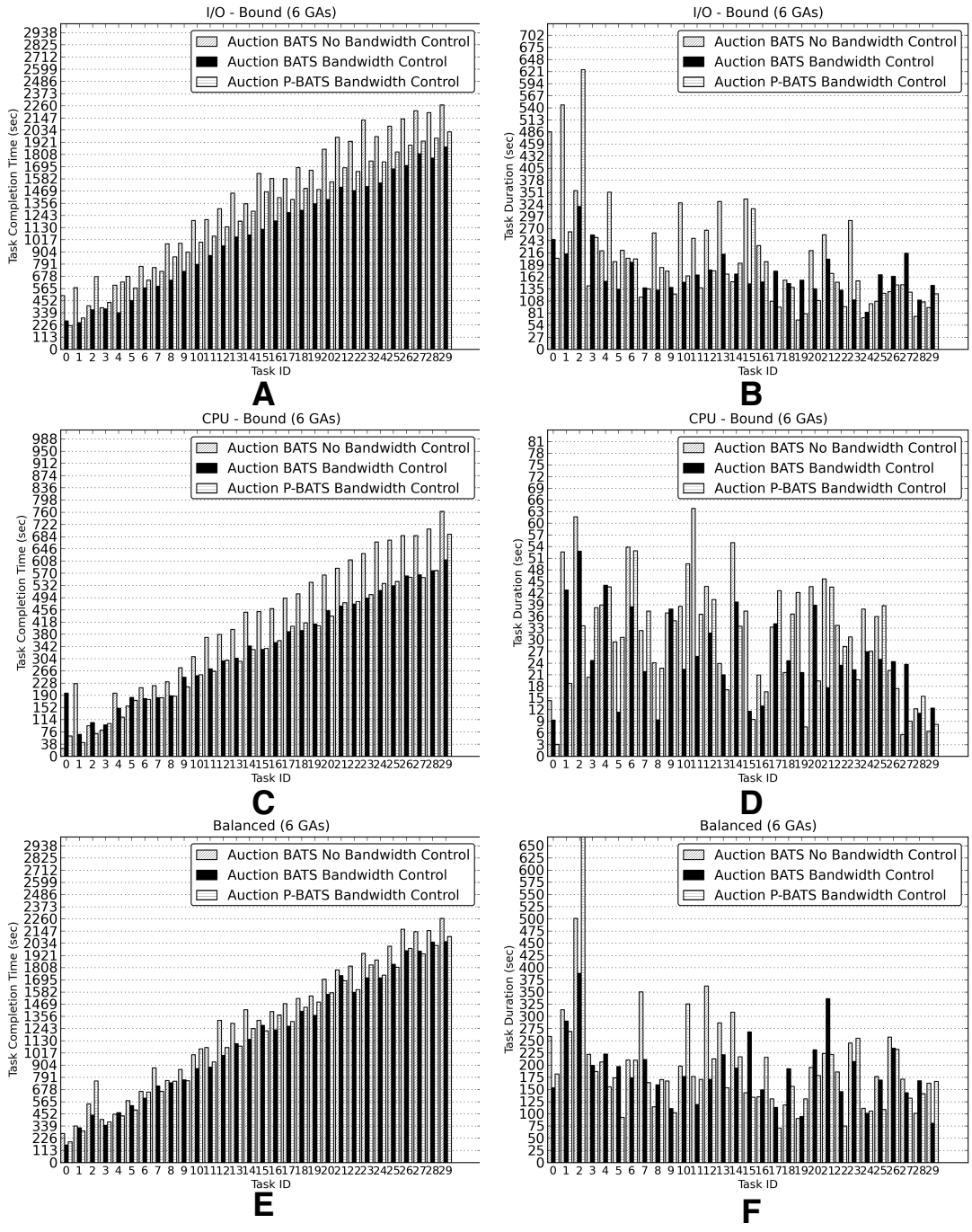


Figure 5.28: Effects of mixed job types on task completion time and task duration (6GAs).

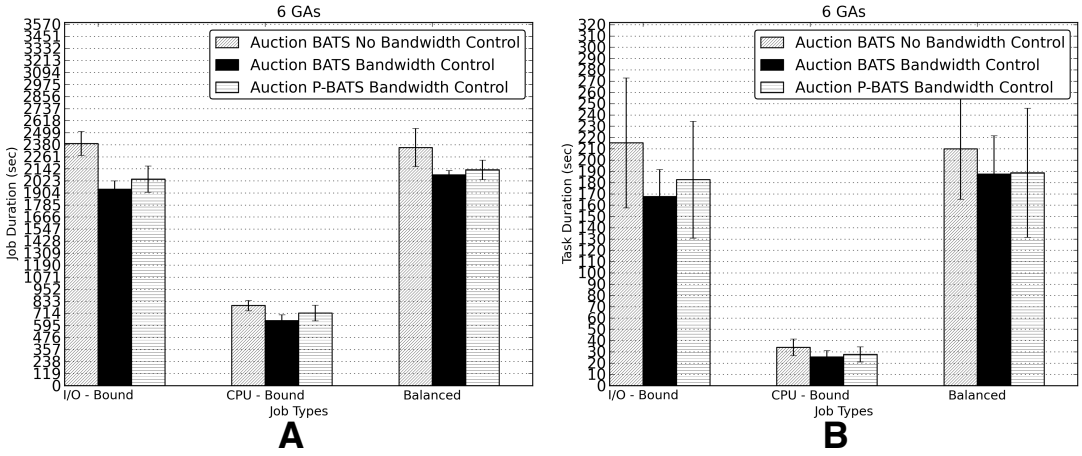


Figure 5.29: Effects of mixed job types on job duration and task duration (6GAs).

progressively block their execution.

5.5 Summary of Results

We are now in a position to summarise the most interesting findings drawn from our empirical experimental study, as follows:

- In heterogeneous experimental settings, exposed to diversified network traffic and job types distribution, considerable performance improvements are achieved whenever heuristic mechanisms, combining TEs computational and local network information, are integrated and applied to the load distribution algorithms. Hence experimental results show that computationally intensive applications, that must be collaboratively executed, achieve faster completion times (i.e. decreased *task completion time*, *job duration* and *task duration* values), whenever informed decisions about heterogeneous traffic contention levels are accounted for in deciding towards which TE nodes to distribute load.

- As expected, *task completion time*, *job duration* and *task duration* decrease by increasing the number of the TEs, while keeping a constant GA number, and increase by increasing the number of GAs, while keeping a constant TE number. Moreover, latencies generally increase by increasing the number of I/O bursts required to be performed to allow remote task computation. In particular, longer latencies can be observed for I/O-bound and balanced jobs. Latencies are, on the other hand, shorter for CPU-bound jobs, since a limited amount of message exchanges is involved during their computation. The same experimental pattern is reflected in the performance gap between the two situations with and without bandwidth control. In fact, greater performance improvements, resulting in pronounced gaps, are seen whenever either I/O-bound or balanced jobs are distributed accounting for TEs local bandwidth information against the case in which traffic details are disregarded. As one would expect, whenever I/O-bound or balanced jobs are required to be collaboratively distributed in an environment affected by considerable network traffic, it be-

comes crucial to make informed decision to equally balance the network load to achieve better performance, hence avoiding further congestion in already contended areas.

- The simpler, less sophisticated, BATS heuristic outperforms, in most of the cases, the more sophisticated P-BATS. This might occur because, in dealing with more complex computations, the latter heuristic might incur the risk of making decisions based on out-of-date information.

- The comparison between experimental results and a theoretically computed lower-bound shows how the latter seems to significantly outperform the former. Such a gap is representative of the considerable impact of radio communication interference, message collision and retransmission mechanisms existing in the reality of network traffic not simulated within the computations of the lower-bound. This behaviour is, in fact, mainly displayed whenever either I/O-bound or balanced jobs must be distributed.

- Experimental results show that the simplicity and intrinsic robustness of the simpler Auction algorithm leads to results outperforming the Lookup List algorithm for heterogeneous, as well as homogeneous, scenarios. This behaviour might again be justified by the intrinsic capability of the Auction algorithm to handle decisions based on more up-to-date network information. Experimental evaluation confirms the trends in results already illustrated in Chapter 4.

5.6 Discussion

In this chapter, we performed an experimental analysis investigating the impact of environmental network traffic contention in distributing heterogeneous applications types. In particular, we characterised the set of heterogeneous applications according to both their computational and communication perspective, namely I/O-, CPU-bound and balanced job categories. We devised an algorithm computing a theoretical lower-bound representing the view of an oracular system capable of taking, in each instant of time, the best offload decisions. Moreover, we defined an additional P-BATS heuristic to measure the impact of distributing heterogeneous kinds of applications within the environment and we compared it against BATS and the theoretical lower-bound. We then tested the DWAG paradigm by distributing the defined heterogeneous applications and applying the set of specified heuristics with actual computation, actual profiling of the medium, and actual network traffic for a multiplicity of settings. Experimental results showed: (i) the strong benefits brought by accounting for local network information in performing job collaborations, especially in distribution I/O-bound and balanced job types; (ii) whenever jobs are mainly computational and require very little message exchange, it may be convenient, if possible, to perform local computations in order to avoid incurring into distribution algorithms overhead; (iii) the advantages brought by simpler heuristic (BATS) against more sophisticated ones (P-BATS).

Chapter 6

Localisation Case Study

In Chapters 4 and 5, we have undertaken an experimental analysis investigating the impact of accounting for local network conditions on distributing computationally intensive applications through DWAG. We have thus analysed the effects of applying a variety of heuristic mechanisms within the load distribution protocols on both homogeneous and heterogeneous network traffic configurations and job profile characterisations. The objective of this chapter is thus to apply our DWAG approach with BATS mechanism to a real-world case study.

The remainder of this chapter is organised as follows: (i) in Section 6.1, we describe the motivation behind the selection of the localisation approach [Li and Kunz, 2009] as a case study for our work; (ii) in Section 6.2, we present the rational behind the selected CCA-MAP localisation scheme; (iii) in Section 6.3, we apply the DWAG paradigm with BATS scheme to the CCA-MAP localisation algorithm, thus we match the algorithmic requirements over the DWAG actors and we describe the additional algorithmic adaptations required to allow the CCA-MAP approach, evaluated exclusively through simulations, to be deployed on real-world testbeds; and (iv) in Section 6.4, we measure the impact of network conditions within the CCA-MAP localisation scheme and we summarise the experimental results.

6.1 Driving Motivation

In this chapter, we describe the application of DWAG with BATS mechanism to a case study, significant for our scenario: that of node localisation.The Self-healing Autonomous Sensing Network (SASNet) presents an advanced Wireless Sensor Network that aims to enhance the effectiveness of mission operation in the contemporary military environment, by providing relevant and accurate situational awareness information. In order to achieve this objective, precise location information is required to be computed in almost real-time in SASNet. We thus selected the existing localisation algorithm described within SASNet and presented by Li et al. [Li and Kunz, 2009, Li, 2008, Li and Kunz, 2007a, Li and Kunz, 2007b] and we adapted it by integrating the DWAG paradigm and BATS heuristic. Localisation as case study has been chosen for several reasons. Firstly, we challenged our approach against a real-world problem recurrent within emergency scenarios. In fact, as described in Section 1.1, whenever an emergency occurs within indoor environments in which GPS support cannot be assumed, it is crucial for external *Command & Control* (C^2) to monitor the position of first responders while they move and act. Secondly, recent

research in the area of node localisation within military and tactical WSNs has promoted the adoption of smart localisation algorithms able to perform fast computations in a distributed fashion directly in-field on devices instead of relying on the presence of both centralised authorities and super-peer nodes specifically elected to perform bulkier computations.

Li et al. [Li and Kunz, 2009]'s algorithm has been adopted as case study within the current dissertation for the following reasons: (i) it is inherently distributed, and thus supports parallel work; (ii) it is challenging from both a computational and a communication point of view, hence it can benefit from node collaborations; (iii) it is specifically envisioned for mission-critical operational network scenarios (i.e. it is time sensitive, since out-of-date location information is of limited use within tactical WSNs); (iv) it is flexible and robust, without requiring additional hardware (i.e. directional antennas) or assuming that sensor nodes already have an estimate of each other locations; and (v) it represents a recent research perspective demonstrating that cooperative localisation schemes can often produce accurate results using a very small number of anchor nodes or even no anchor nodes.

Moreover, the work presented in this chapter is the result of a collaboration between University College London, UK and the Communications Research Centre, Canada. The aim of such collaboration has been twofold: (i) the practical development on real hardware and testbeds of a theoretical approach so far exclusively evaluated through Matlab simulations; (ii) the adoption and integration of our DWAG paradigm distributing load through an Auction algorithm enriched with BATS strategy within the existing localisation approach in order to manage distribution. In fact, while simulations represent a straightforward way to evaluate and compare theoretical approaches because of their extreme flexibility, modularity, scalability and adaptability, they make many simplifying assumptions, especially regarding radio communication behaviours, which are instead crucial whenever distribution plays a role in supporting application execution.

6.2 SASNet Localisation Algorithm Overview

We now provide an overview of the localisation algorithm presented in SASNet and devised by Li et al. [Li and Kunz, 2009].

In their work, the authors propose a cooperative node localisation scheme applying a non-linear data mapping (NLM) technique, the *Curvilinear Component Analysis* (CCA) described in Section 6.2.1, to produce accurate node position estimates for tactical WSNs. In particular, the CCA is a technique from neural networks adopted by [Li and Kunz, 2009] to exploit the learning ability of nodes to self-organise into maps of coordinates. The computed maps are then disseminated through a distributed CCA-MAP algorithm, detailed in Section 6.2.3, capable of deriving node locations in either range-based or range-free scenarios. The advantages of this localisation algorithm are related to the fact that it requires only a minimum number of anchor nodes to convert relative position maps into absolute location values.

In their work, the authors compare the validity of their approach against another leading cooperative node localisation algorithm, namely MDS-MAP, which employs Multi-Dimensional Scaling (MDS) techniques to achieve node localisation. From this comparison, they demonstrate that the CCA-MAP approach significantly improves position estimate accuracy in many of the scenarios they simulate. How-

ever, in the following evaluation we are not interested in measuring the localisation accuracy in terms of generated location maps. Instead, we tackle Li et al. [Li and Kunz, 2009]'s approach with the aim of understanding its behaviour, of adapting it to be practically implemented on deployed testbeds, of investigating communication issues arising from implementing a distributed approach exclusively evaluated through simulations, in reality and of measuring the benefits brought from the application of our BATS mechanism in handling DWAG distributions.

6.2.1 Curvilinear Component Analysis

We begin with a high level description of the grounding principles underlying the CCA-MAP localisation approach. In their work [Li and Kunz, 2009], Li et al. adopt a non-linear mapping method, named Curvilinear Component Analysis (CCA) [Demartines and Herault, 1997], to allow each node in the system to compute the map of its neighbouring nodes, as follows.

Given N input vectors $\{x_i, i = 1, ..., N\}$ where each vector x_i is of n input dimensions (e.g. ranges between all nodes), CCA looks for N output vectors $\{y_i, i = 1, ..., N\}$ where each y_i is of s dimensions $(s < n)$ (e.g. absolute position in 2 or 3D space). The relative distance between input vector x_i and x_j is preserved between y_i and y_j. That is, given the distance between x_i's as $X_{ij} = d(x_i, x_j)$ and the corresponding distance in the output space $Y_{ij} = d(y_i, y_j)$, CCA pushes Y_{ij} to match X_{ij} for each possible pair (i, j) while minimising the cost function in Equation 6.1.

$$\mathrm{E} = \frac{1}{2} \sum_i \sum_{j \neq i} (X_{ij} - Y_{ij})^2 F(Y_{ij}, \lambda_y) \tag{6.1}$$

$F(Y_{ij}, \lambda_y)$ in Equation 6.1 represents a weighing function often chosen as a bounded and monotonically decreasing function to favour local topology conservation as in Self-Organising Maps (SOM). Decreasing exponential, sigmoid, or Lorentz functions are all suitable choices for such function F.

The computing efficiency of CCA emerges in the minimisation process of the cost function in Equation 6.1. In fact, compared to other methods such as the stochastic gradient descent or the steepest gradient descent where one vector y_i is moved every time according to the sum of every other y_j's influence, CCA temporarily pins one y_i and moves all the other y_j around, without regard to interactions amongst the y_j. The update for each cycle is then defined in the way described in function in Equation 6.2.

$$\Delta y_j = \alpha(t) F(Y_{ij}, \lambda_y)(X_{ij} - Y_{ij}) \frac{(y_j - yi)}{Y_{ij}} \forall j \neq i \tag{6.2}$$

The parameter $\alpha(t)$ in Equation 6.2 decreases with time, as usual for stochastic gradient methods. This rule for update in each cycle is much simpler than stochastic gradient as only the distances from node i to the others need to be computed, instead of all the $\frac{N(N-1)}{2}$ distances in both the input and output spaces. For an adaptation cycle of all nodes (except i), the complexity is $O(N)$ instead of $O(N^2)$. This not only converges the computation much faster, but also makes it more likely to eventually escape from local minima of E [Demartines and Herault, 1997] in Equation 6.1.

6.2.2 CCA Algorithm

The CCA technique is thus applied in Algorithm 6 to allow each node in the system to compute a *local map* of node coordinates in its own reference system containing both its coordinates and those of its neighbours. Therefore, given the main CCA principles and using distance constraints, the localisation problem can be stated as follows.

Given a distance matrix $D_{N\times N}$ of N nodes, find the coordinates of all the points to minimise the function in Equation 6.3.

$$\min \sum_{ij}(d_{ij} - p_{ij})^2 i, j = 1, ..., N \tag{6.3}$$

In particular, d_{ij} and p_{ij} values in Equation 6.3 are described as follows:

- d_{ij} is the either measured or known distance between node i and j;
- p_{ij} is the distance between node i and j computed using the calculated coordinates of i and j.

If d_{ij} is taken as the distance matrix of the input data set and p_{ij} the distance matrix of the output data set, CCA then pushes Equation 6.3 to a minimum as it minimises the cost function in Equation 6.1.

Being the distance matrix $D_{N\times N} = (d_{ij})_{N\times N}$ the only known data of these N nodes, it is used as both the input data set (i.e. $x_{N\times N} = D_{N\times N}$) and the inter-vector distance matrix of the input data set (i. e. $X_{ij} = D_{N\times N}$). Even though $D_{N\times N}$ is not the real distance between vectors (i.e. the row vectors) in $D_{N\times N}$, the CCA algorithm projects data points quite well given a defined distance matrix without requiring that it is the real Euclidean distance between the input data vectors.

The CCA data reduction preserves the distances between every two data points in the input data space while generating the output data set (with each data point having a reduced dimension). Thus, taking the distance matrix $D_{N\times N}$ of N nodes as the input data set (i.e. $x_{N\times N} = D_{N\times N}$), each vector in the input space is of N dimensions. CCA reduction is applied so that the output data set contains the N vectors each reduced to a dimension of 2, or 3, according to a 2D, or 3D, dimensional space reduction, respectively. The authors focus on a 2D space within their discussions. Unlike methods such as iterative trilateration, cooperative algorithms using NLM often incur very little extra cost computing positions in 3D space compared with that in 2D. This property makes the NLM based cooperative algorithm advantageous for real network deployments. The output data set is denoted as y_{Nx2} which represents in fact the 2D coordinate matrix of the N nodes. The inter-vector distance in the input data space is thus forced to be $X_{ij} = D_{N\times N}$ to push the inter-point distance in the output data space to $Y_{ij} = D_{N\times N}$, even though $D_{N\times N}$ is not the real distance between vectors in the input data set $x_{N\times N}$.

The CCA procedure consists of the steps described in the Algorithm 6.

The authors select $F(Y_{ij}, \lambda) = e^{\frac{Y_{ij}}{\lambda(t)}}$. Both $\lambda(t)$ and $\alpha(t)$ decrease with time within each computing cycle c. In particular, the function in Equation 6.5 was adopted within the simulations to implement $\lambda(t)$ and $\alpha(t)$ (i.e. $\lambda(t) = \nu$ and $\alpha(t) = \nu$), though other similar functions can be selected instead. Thus, c is the number of total computing cycles, also called the CCA training length.

Algorithm 6: CCA Algorithm

/* **Initial setup of output** y_{Nx2}**:** */
foreach *(node)* **do**
  compute the mean values of the first two columns of the input data set $x_{N\times N}$;
  adjust these values by a uniformly randomised standard deviation of the same column;
  set y_{Nx2} with the computed initial output estimation values;
end
/* **Mapping Node Coordinates through CCA:** */
for *(c computing cycles)* **do**
  repeat
    randomly select node i;
    foreach *(node* j $(j \neq i)$*)* **do**
      compute the new $y_j(t+1)$ from the current value of $y_j(t)$ through Equation 6.4;

$$y_j(t+1) \quad = \quad y_j(t) + \alpha(t) e^{\frac{-Y_{ij}}{\lambda(t)}} \left(\frac{X_{ij}}{Y_{ij}} - 1\right)(y_j - y_i) \tag{6.4}$$

    end
  until *(all nodes have been selected)* ;
end

$$\nu(t) \quad = \quad \nu(0) \times \left(\frac{\nu(c)}{\nu(0)}\right)^{\frac{t}{c-1}} \tag{6.5}$$

The number of training cycles c required is related to the size of the input data set and also the accuracy of the distance matrix. The bigger the input data set (i.e. the larger N), the fewer the cycles are required in projecting the final output data. A more accurate distance matrix would also result in fewer cycles, as the cost function in Equation 6.1 will decrease much faster towards the minimum. A maximum number may be assigned to the total cycles allowed in the algorithm. During the execution, if the cost function $E < \epsilon$ before reaching the maximum number of cycles, the algorithm exits and the projected data form the final output data set.

Though the above description applies CCA to the task of determining node locations, the distance matrix used as the input for the algorithm is often not available for large networks. Recently, Drineas et al. [Drineas et al., 2006] proposed algorithms for distance matrix reconstruction for sensor network localisation using single value decomposition. This may provide an option to obtain the distance matrix for the network. However, in their work the authors assume that the distance matrix of the network is unknown. Instead, a distributed map algorithm [Demartines and Herault, 1997] is adapted in the scheme to compute the node coordinates in the network.

After performing Algorithm 6, each node has therefore a *local map* containing both its coordinates and those of its neighbours in its own reference system.

6.2.3 CCA-MAP Algorithm

Adopting steps similar to the distributed map MDS-MAP algorithm [Shang et al., 2003], the authors propose in their work [Li and Kunz, 2009] an alternative distributed CCA-MAP algorithm. In particular, the CCA-MAP scheme similarly to MDS-MAP builds *local maps* for each node in the network and then merges them together to form a *global map*. Unlike MDS-MAP, CCA is employed in computing the

node coordinates in the *local maps*.

Once each node has computed a *local map* of node locations, for itself and its neighbours in its own reference system through the CCA algorithm by using the local information only, such maps are collaboratively patched together in order to form a *global map*. CCA-MAP performs this role.

If an accurate ranging capability is available in the network, the local distance between each pair of neighbouring nodes is measured and known. Otherwise, connectivity information is used to assign value 1 to the edge between each neighbouring pair of nodes. From this, a distance matrix for all the nodes in the R_{lm} (e.g. $R = 2$) hop neighbourhood of node x can be constructed using the shortest distance matrix as an approximation. Instead of a fixed $R_{lm} = 2$, as that used in MDS-MAP [Shang et al., 2003], R_{lm} can be adjusted in CCA-MAP. The CCA reduction technique can generate accurate results with a reasonably accurate distance matrix of a small size, for example, a distance matrix of 12 or bigger. Therefore, in the range-based scenarios where the local distance measurements are known with a certain level of accuracy, the one hop neighbourhood distance matrix of a certain size (e.g. size $> 12 \times 12$) not only is more accurate than the two hop distance matrix of approximation, but can also be computed faster using CCA to produce more accurate position estimates. In the range-free options, a bigger size of the distance matrix assists better in determining the node position coordinates using CCA. Thus in the range-based option of the algorithm, for any given node, if its one hop neighbourhood has more than 12 nodes, $R_{lm} = 1$ may be chosen for improved performance. Otherwise, $R_{lm} = 2$ is applied. In the range-free computations of the *local map* where the local distance matrix is particularly inaccurate using the hop count approximation, often $R_{lm} = 1$ is only taken when the one hop neighbourhood expands to a much larger size of 30 or 40.

Algorithm 7: CCA-MAP Algorithm

```
1 foreach (node) do
2     build the local map by filling the connectivity matrix including neighbouring nodes within
      R_lm hops;
3     compute the shortest distance matrix of the local map and take it as the approximate
      distance matrix LD;
4     apply the CCA procedure to generate the local map (Algorithm 6);
5     patch local maps of different nodes to generate a global map of locations (Algorithm 8);
6     transform the patched global map into an absolute map based on the absolute location of a
      sufficient number of anchor nodes;
7 end
```

As described in Algorithm 7, each node first fills the connectivity matrix including neighbouring nodes within R_{lm} hops. Then, it computes the shortest distance matrix and it applies the CCA procedure (Algorithm 6) to generate a *local map* containing its coordinates and its neighbours coordinates in its own reference system. Once each node has computed its own *local map*, such maps need to be progressively patched together through an iterative procedure.

The computation of a *global map* of node locations is performed as described in Algorithm 8. As illustrated in Figure 6.1, initially a node N is randomly selected and its *local map* is chosen as starting current map $N_{LocalMap}$. Then, N selects from among its neighbouring nodes that (e.g. M) sharing most

Algorithm 8: Patch Algorithm

foreach *(randomly selected starting node N)* **do**
  select N's *local map* as the starting current map ($N_{LocalMap}$);
  select N's neighbouring node M sharing most neighbours with N;
  request $M_{LocalMap}$;
  sort $N_{LocalMap}$ and $M_{LocalMap}$ to match coordinates of common neighbouring nodes;
  compute SVD transformation matrices using coordinates of common neighbouring nodes;
  apply generated SVD matrices to patch $M_{LocalMap}$ into $N_{LocalMap}$'s own reference system;
end

neighbours with it. N requests M's *local map* (i.e. $M_{LocalMap}$) and it sorts its own map $N_{LocalMap}$ and M's map $M_{LocalMap}$ in order to match coordinates of common neighbouring nodes. N then performs a linear transformation using the coordinates of the nodes common to both maps. The output of such transformation is finally adopted to patch $M_{LocalMap}$'s coordinates into $N_{LocalMap}$'s coordinates in N's own reference system, thus generating a *global map*. The procedure is iterated until each node has computed a *global map* in its own reference system. Each *global map* is then converted into an *absolute map* of locations whenever a sufficient number of anchor nodes is deployed in the system (i.e. ≥ 3 for 2D space and ≥ 4 for 3D).

Singular Value Decomposition (SVD) has been selected as transformation procedure. Thus, to merge $M_{LocalMap}$ into $N_{LocalMap}$, a linear transformation involving translation, reflection, orthogonal rotation, and scaling, is applied to ensure that the coordinates of the nodes common to $N_{LocalMap}$ and $M_{LocalMap}$ best match each other [Shang and Ruml, 2004, Shang et al., 2004]. Thus, given the coordinates of common nodes in maps $N_{LocalMap}$ and $M_{LocalMap}$ as matrices $X_{N_{LocalMap}}$ and $X_{M_{LocalMap}}$, the linear transformation $T(.)$ delivers the minimum sum of the squared errors (i.e. $\min_T ||T(X_{M_{LocalMap}}) - X_{N_{LocalMap}}||_2$) to merge map $M_{LocalMap}$ into $N_{LocalMap}$. Once the SVD is computed, three transformation matrices are obtained and used to patch a *global map* in N's reference system. In particular, the coordinates of the nodes exclusively belonging to $N_{LocalMap}$ remain unchanged. Those belonging solely to $M_{LocalMap}$ are transformed according to the computed SVD transformation matrices. Finally, for the coordinates common to both node maps, the mean between the original values and the transformed ones is computed. In [Li and Kunz, 2009], the authors state that any node can be potentially selected to perform the merge, though practically, certain nodes in the network that have more computing power or that need to know positions of other nodes can be selected to construct the *global map*.

In the CCA-MAP scheme, unlike the MDS-MAP algorithm, no refinement procedure is applied, since the results are often satisfactory without further optimisation. The computing of the *local map* can be distributed at each local node, or can be carried out at more powerful gateway nodes of each cluster should the sensor network have a hierarchical structure to relieve the severely resource-limited sensor nodes from any of the computing and communication demands imposed by localisation. Moreover, the *local maps* can be merged in parallel in different parts of the network by selected nodes. There is no need for anchor nodes in merging the maps. When at least three anchor nodes are found in the merged map of

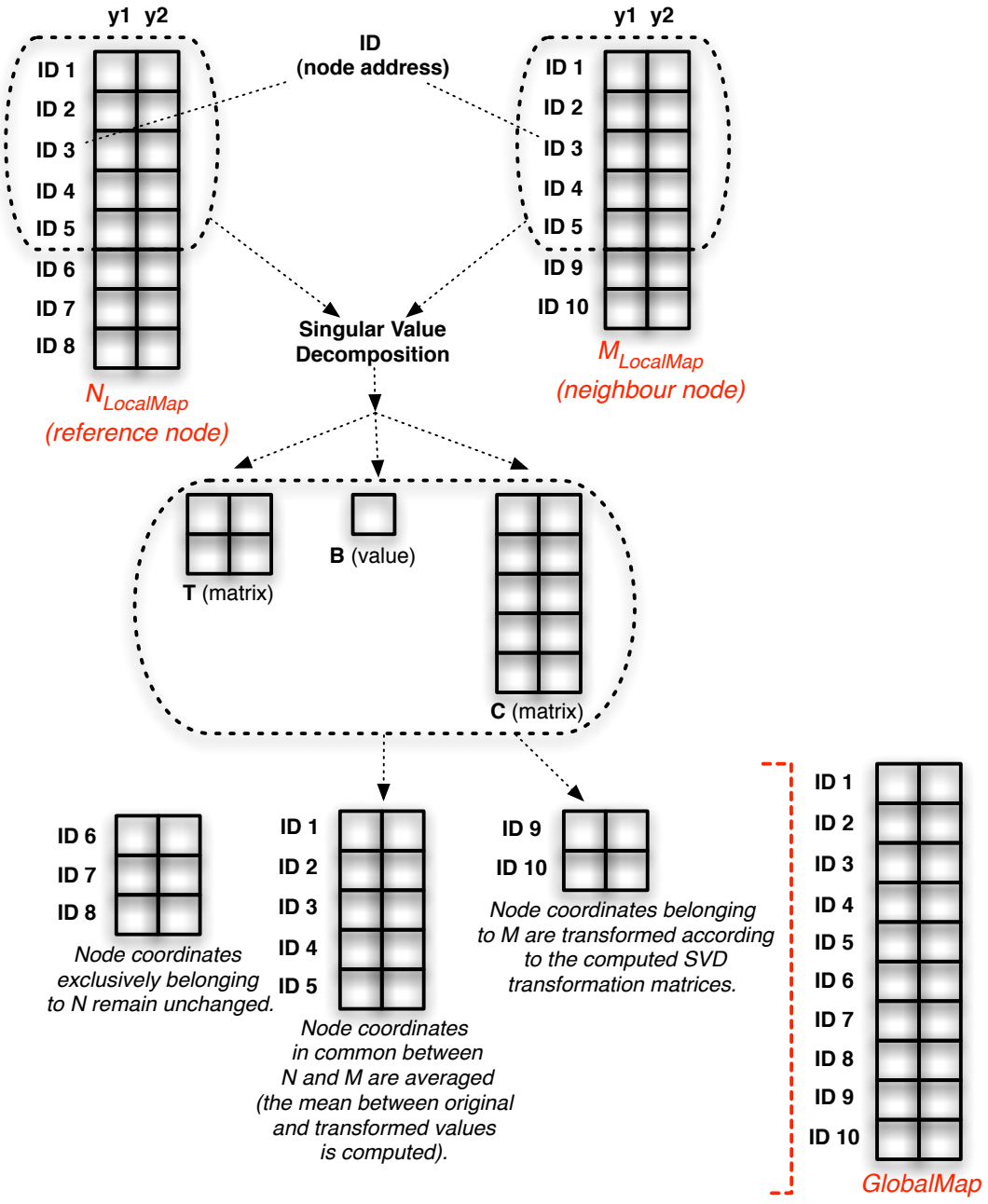


Figure 6.1: CCA-MAP localisation algorithm global map patching procedure.

a subnetwork, an absolute map of the subnetwork can be computed using the coordinates of the anchor nodes to obtain the absolute coordinate values of all the nodes in the map of the subnetwork. Alternatively, the CCA-MAP algorithm can be performed in a completely distributed way all over network and node locations can thus be collaboratively computed by distributing cumbersome computations to specific nodes in the network. Hence the CCA-MAP localisation algorithm represents a case study suitable for testing our DWAG paradigm while investigating the impact of network contention and bandwidth occupancy while performing distributed and collaborative computations.

6.3 DWAG/BATS and CCA-MAP Integration

In Section 6.2, we described the logic behind the CCA-MAP localisation algorithm proposed by Li et al. [Li and Kunz, 2009]. Although in their experimental evaluation the authors emphasise the inherently distributed nature of the CCA-MAP proposed algorithm, they exclusively evaluate it in a centralised way through simulations, without tackling the concrete issues related to distribution within deployed systems.

In this section, we are now ready to describe the application of the DWAG paradigm to the CCA-MAP algorithm. In particular, in Section 6.3.1 we identify the main tasks into which the CCA-MAP job can be split. Further, we match the localisation tasks to the DWAG paradigm actors, GAs and TEs.

In Section 6.3.2, we describe the practical adaptations brought to the original CCA-MAP localisation scheme to allow its fully distributed execution through DWAG. We then list the set of computational and communication issues that must be tackled in moving from simulation to experimental deployments.

6.3.1 Localisation Tasks Identification

We are now in a position to identify the main tasks $Task_{i,j}$ into which we split the localisation Job_i to allow for the application of the DWAG paradigm. Moreover, we match the localisation algorithm requirements and tasks over the DWAG actors, GAs and TEs, as follows:

- GAs drive the main execution flow of the localisation Job_i by performing the following $Task_{i,j}$:

  - $Task_1$ (t_1): the GA broadcasts a message to discover its neighbours and to collect from them their adjacency matrices. This phase was absent from the original algorithm implementation presented in [Li and Kunz, 2009] because of the centralised way in which it was simulated. In fact, this is the preliminary *discovery phase* undertaken within the DWAG paradigm to allow the GA for neighbourhood discovery (Section 4.2).

    Deliverable: the GA establishes its own neighbourhood.

  - $Task_2$ (t_2): the GA computes its own *local map* by applying the CCA procedure (Algorithm 6). In addition, the GA chooses among its neighbours (those stored in its adjacency matrix) the one that shares the greatest number of neighbours with itself, requests and obtains its *local map*.

    Deliverable: the GA has two *local maps* to be patched together.

  - $Task_4$ (t_4): the GA patches the two *local maps* into a *global map* in its own reference system by merging neighbouring node *local map* with its own local one. This is done by adopting

the SVD transformation matrices computed and received back from the elected TE node.

Deliverable: the GA outputs a *global map* containing its own and its neighbours computed locations.

- The TEs are asked to perform the SVD computation and thus the following $Task_{i,j}$ is identified:

  - $Task_3$ (t_3): the TE is selected by the GA for the computation of the SVD transformation matrices. The TE is thus elected to receive two maps from the GA (i.e. two partial *local maps* containing coordinates of common nodes within the original *local maps*) and to perform the SVD transformation on them.

    Deliverable: the TE outputs three transformation matrices that are then delivered back to GA to allow it for *global map* computation.

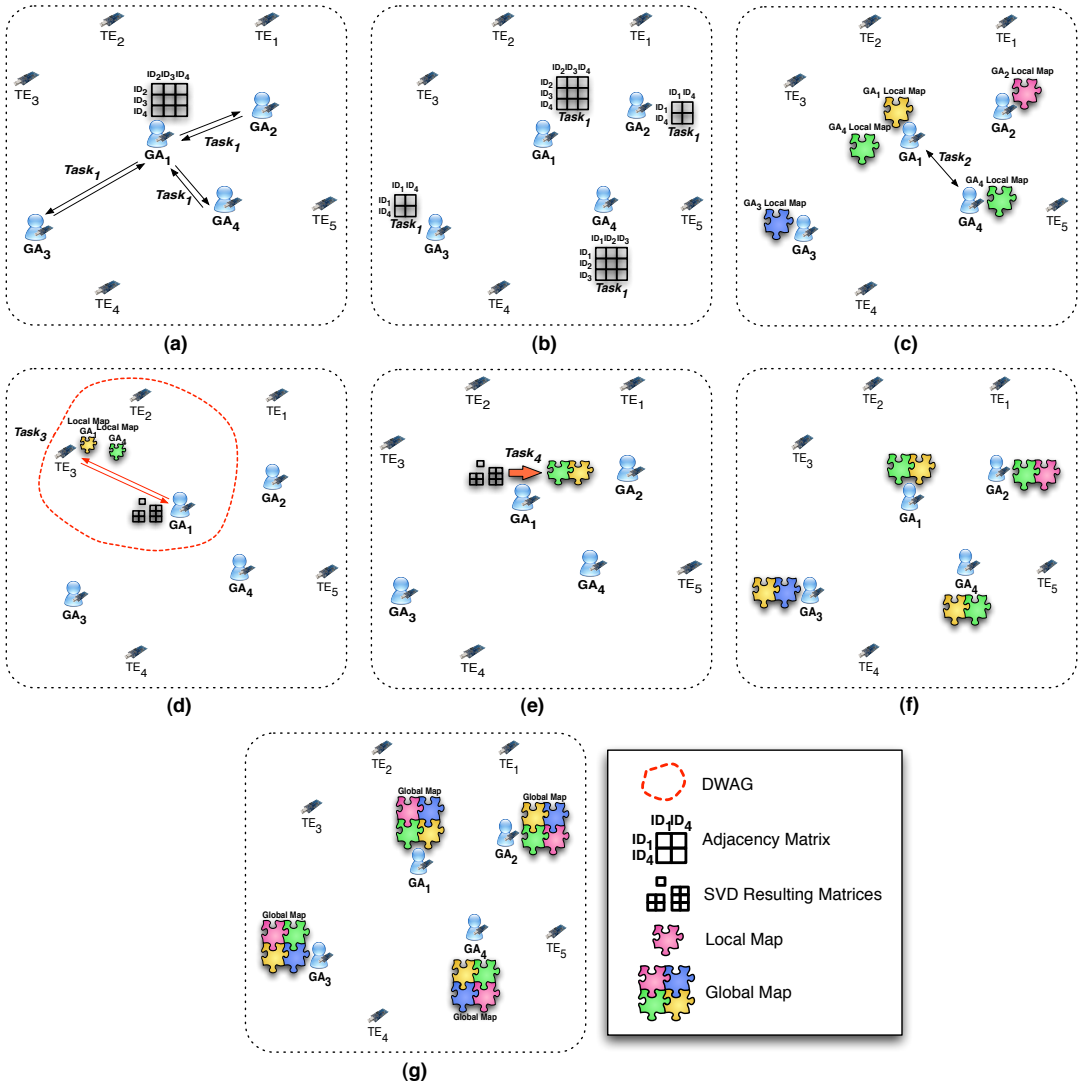


Figure 6.2: Tasks identification within the CCA-MAP localisation algorithm.

Let us now describe the mapping between the CCA-MAP localisation algorithm and DWAG through the example illustrated in Figure 6.2. Firstly, each GA broadcasts a message to discover its

neighbourhood and to collect from its neighbours their adjacency matrices ($Task_1$ in Figure 6.2(a)). At the end of this phase, each GA is aware of its own neighbourhood (Figure 6.2(b)). Secondly, each GA computes its own *local map* by applying the CCA procedure (Algorithm 6). Moreover, each GA chooses among its neighbours (those stored in its adjacency matrix) the one that shares the greatest number of neighbours with itself, requests and obtains its *local map* ($Task_2$ in Figure 6.2(c)). For example, let us follow GA_1's behaviour and let us assume that GA_4 is GA_1's neighbour and also the node that shares the greatest number of neighbours with GA_1. Having computed its own *local map* $GA_{1_{LocalMap}}$, GA_1 asks for $GA_{4_{LocalMap}}$. GA_1 is now in possess of two *local maps* that must be patched together. However, since the patching phase involves a computationally intensive SVD transformation to be executed, GA_1 takes advantage of the DWAG paradigm and creates an ad-hoc grid with local auxiliary TE nodes in order to distribute tasks to carry the burden of heavy computation ($Task_3$ in Figure 6.2(d)). Hence TE_3 is selected by GA_1 for the computation of the SVD transformation matrices. TE_3 is thus elected to receive two maps from GA_1: two partial maps containing coordinates of nodes common to $GA_{1_{LocalMap}}$ and $GA_{4_{LocalMap}}$. Therefore, TE_3 outputs a set of transformation matrices that will be sent back and used by GA_1 to perform the final $GA_{1_{LocalMap}}$ and $GA_{4_{LocalMap}}$ patch into a unique $GA_{1_{GlobalMap}}$ ($Task_3$ in Figure 6.2(e)). The procedure is iteratively repeated by every GA (Figure 6.2(f)), until each of them builds a *global map* of the system nodes in its own reference system (Figure 6.2(g)). Relative maps are converted into absolute maps in the presence of anchor nodes. We do not deal with anchor nodes within our work, and thus we aim only at building a global relative patched map.

6.3.2 Moving from Theory to Practice

We now describe the main practical adaptations brought to the original CCA-MAP localisation scheme to allow its fully distributed execution through DWAG collaborations.

Firstly, note that all the original simulations [Li and Kunz, 2009] carried on for the CCA-MAP evaluation were run in a centralised way using Matlab V7.2 on a 1.60GHz Pentium M processor with 1GB RAM. In this work, we implement the complete CCA-MAP localisation scheme in a fully distributed manner, exploiting DWAG collaborations as described in Section 6.3.1, on TMote Sky [TELOSB, 2010, Sentilla, 2010] devices, each of which has a MSP430 8MHz 16-bit micro-controller, a ChipCon CC24202 radio module supporting IEEE 802.15.4 (with 250kbps as maximum raw bandwidth), 10kB RAM and 48kB Flash ROM. We thus used devices that are $O(200)$ (i.e. $\frac{1600}{8}$) times slower and 100,000 (i.e. $\frac{1{,}000{,}000}{10}$) times less capable than a common computer.

Effect of constrained memory resources: One of the first problems, met by developing the CCA-MAP algorithm on a real testbed, was a memory problem. When working with simulators running on standard processors (e.g. Pentium), floating point libraries do not usually represent an issue since the hardware processor already supports them. On the other hand, the same assumptions might not hold for resource-scarce micro-controllers that might thus not support them by default. However, since floating points are used within localisation algorithms to test location accuracy, we nevertheless implemented an external library emulating floating point computation, with the side effect of occupying precious memory resources at run-time. This is illustrative of the observation that no matter how much computational

power the actual technology squeezes inside an incredibly constrained form factor, a tradeoff between limited size and computational capability will always represent an issue, since small devices (e.g. ideally the size of a dust particle) will never be able to individually perform all the computations required to handle the whole application complexity.

Algorithm 9: Neighbourhood Discovery Algorithm

```
/* GA_List = {{ID_1; ID_1List}, ..., {ID_N; ID_NList}} contains GA's neighbours
   ID_i and the list of their neighbours ID_iList.                          */
/* GA Neighbourhood Discovery:                                              */
1  broadcast a neighbourhood discovery request;
2  repeat
3      if (ID_i ∈ GA_List) then
4          if (ID_iL ≠ ID_iList) then
5              update ID_iList with ID_iL;
6              broadcast GA_List update;
7          end
8      end
9      else if (ID_i ∉ GA_List) then
10         insert {ID_i; ID_iL} in GA_List;
11         broadcast GA_List update;
12     end
13 until (receiving reply {ID_i; ID_iL}) ;
14 if (receiving neighbourhood discovery request from other GAs) then
15     send reply with updated {GA_ID; GA_List};
16 end
```

Let us now describe task $Task_1$'s behaviour. In fact, in order to allow a practical deployment of the simulated [Li and Kunz, 2009] CCA-MAP scheme, we introduced $Task_1$, and thus the initial neighbourhood discovery phase, to the localisation algorithm. While in a simulated approach it is always possible to assume the presence of an adjacency matrix that is as large as required filled with either 0/1 node connectivity information (i.e. range-free scenario) or local distances (i.e. range-based scenario), such starting point values are not available by default and must be fetched through active node collaboration. Hence a neighbourhood discovery algorithm (i.e. $Task_1$) has been added to the GA's side in the original CCA-MAP algorithm. Details are presented in Algorithm 9. Initially, each GA broadcasts a neighbourhood discovery request to (i) assert the GA peers' ID_i within its communication range, and (ii) obtain their adjacency matrices ID_{i_L}. This action is simple whenever a single GA alone is required to discover its own neighbours, since it broadcasts a request and waits for neighbours replies. However, the action becomes more challenging when multiple GAs simultaneously boot and suddenly start broadcasting messages. Consequently, we adopted random Contiki OS timers to desynchronise message emission and thus to limit possible message collisions. Secondly, we introduced the mechanism presented in Algorithm 9. Whenever GA receives a broadcast reply from a node ID_i which does not belong yet to its own adjacency matrix GA_{List} (i.e. $ID_i \in GA_{List}$), or whose adjacency matrix ID_{i_L} does not match the previously received one $ID_{i_{List}}$ (i.e. $ID_{i_L} \neq ID_{i_{List}}$), the GA initiates a further broadcast, so that for each node added later to the group a broadcasting wave is triggered in its immediate proximity. In this way, GAs are constantly updated with both their neighbours and their neighbours's

adjacency matrices. This phase may look expensive in terms of communication, but the traffic peak is only reached when nodes are initially inserted into the system. Therefore, this procedure (i.e. $Task_1$) is performed only as an offline bootstrap procedure, not to be iterated during the execution of CCA-MAP. Finally, notice that during $Task_1$ GA can be required either to actively discover its own neighbourhood or to passively provide its own details GA_{List} to GAs interested in building their own adjacency matrix, respectively.

Once the neighbourhood discovery phase is completed, each GA thus has a list (i.e. GA_{List}) filled with its own neighbours (i.e. ID_i) and, for each of them, their adjacency matrices (i.e. ID_{i_L}). From a practical perspective, we stored within GA_{List} not only the adjacency information but also the Received Signal Strength Indication (RSSI) value that each node has with respect to its neighbours. This has been done to test the computational part of the CCA-MAP scheme, thus the computation of the distance matrix in $Task_2$, using values different from those adopted within a pure range-free adjacency matrix (i.e. based on node connectivity information only). However, it is crucial to recall that within this dissertation we are not primarily interested in evaluating the accuracy of localisation results, thus we made the aforementioned choice to better test nodes' computational capabilities.

Another issue emerging from the real deployment of the CCA-MAP scheme is the SVD linear transformation (i.e. $Task_3$). While in a simulated environment SVD is relatively trivial, such computation represents a demanding assignment whenever it must be executed on resource-constrained devices. In particular, to cope with the scarcity of memory resources and with demanding computation, we distribute $Task_3$ to the auxiliary TE nodes deployed in the system.

Effect of radio communication: As detailed in Section 2.2.2, besides computational issues, the strongest assumptions implicit in approaches exclusively evaluated within simulation environments mainly regard radio communication.

In their work [Li and Kunz, 2009], Li et al. assume perfect communication. In fact, they attempt to determine the efficiency of a distributed algorithm designed for a very practical tactical military-based scenario by ignoring the impact of the communication generated by the algorithm itself. However, especially in emergency situations, we can neither assume the presence of an ideal, congestion free, environment nor can we discard the impact of such communication overhead on the performance of the algorithm.

While performing our experiments, we experienced the same radio communication issues discussed in Section 2.2.2, e.g. different kind of interference and message collisions generated both at physical and MAC layers. Therefore, to cope with message collisions and consequent retransmissions, additional control code was implemented to check the integrity of the exchanged localisation maps and matrices. Such extra code adds itself to the actual computational load of the implemented algorithms. This highlights how communication indirectly affects computation and how difficult it is to provide a complete performance evaluation exclusively within simulators.

6.4 CCA-MAP Evaluation with DWAG/BATS

We now detail the set of experiments undertaken within deployed systems to measure the impacts of traffic network congestion on system performance while performing collaborative computation.

Experimental data was gathered from experiments performed within the UCL-CS HEN testbed, whose details were provided in Section 4.6.2. Moreover, experiments were performed according to the assumptions listed in Section 1.4. Section 6.4.1 presents the experimental setup, while Section 6.4.2 describes the obtained results.

6.4.1 Experimental Setup

We now detail the experimental setup adopted for the evaluation of the localisation case study.

In our tests, we used different topological sets of 15 nodes belonging to the UCL-CS HEN testbed, because this is usually the number either of first responders taking part in a single search and rescue operation, or of soldiers collaborating with each other within a unit of Military Operation on Urban Terrain (MOUT) [Desch, 2001]. Furthermore, the results obtained could easily be scaled to bigger networks with a similar degree of connectivity, since the locally generated traffic would be the same. In particular, for each test we selected uniformly distributed network topologies of up to 10 GAs, 4 TEs and 1 interferer node Str injecting background traffic.

The interferer node Str emulates the presence within the environment of diversified computing tasks and communications that are potentially carried out in parallel within the WSN while the highly-demanding localisation computation is performed. This was done both to mimic a realistic and heterogeneous background level of communication, and to inject further real network traffic into the system, in order to study the effects of high network congestion while allowing experimental repeatability.

We set the radio transmission power of Str to the second lowest possible level (i.e. -28 dBm in [Instruments, 2006]) to congest only part of the overall area, reproducing a situation like that described in Section 4.6.1. Thus, Str is located within the radio range of only a subset of TE nodes. However, this value represents an approximation since, because of the vagaries of the radio medium, nodes belonging to the uncongested part of the area might also suffer from slight interference. As for the experiments described in Sections 4.6 and 5.4, Str was configured so that bursts of 3 bytes were injected at regular intervals after each time a trivial computation was executed.

Again, we programmed TMote Sky devices by setting the Radio Frequency (RF) ZigBee channel to 26 in order to minimise the radio interferences between IEEE 802.15.4 ZigBee and IEEE 802.11b WiFi.

In the experiments, we used actual computation and actual network traffic. Three communication phases are included in our experiments during task distribution: (i) offload of the data (but not code) associated with tasks from the GA to the TE; (ii) in-progress communication exchanges between the GAs and the TEs needed to progress task execution; (iii) upload of results from the TE to the GA. The CCA-MAP localisation scheme has been implemented over the testbed using DWAG mappings, as described in Section 6.3.1. Thus, the GAs drive the main execution flow of the localisation case study Job_i that has been split into a set of tasks $Task_{i,j}$ (i.e. $\{t_1, \ldots, t_4\}$). In particular, some of the tasks are directly executed on GAs (i.e. $\{$t1,t2,t4$\}$), while the execution of the most computationally intensive is

distributed to the auxiliary TEs (i.e. $\{t_3\}$) through DWAG.

Job duration is used as performance metric in the experiments since timeliness of information is vital in tactical scenarios. Moreover, latency captures the effects that tasks from one node have on the execution patterns of others. One may argue that battery life is an important metric. However, in the tactical scenario, battery lifetime is much less of an issue than the timeliness of information. In fact, information from sensors will be most useful within the first few tens of minutes of an incident. In the experiments, sensors were able to run the algorithms while battery powered for more than 48 hours between charges, thus fulfilling the battery requirements of our scenario. Since within the case study only one kind of task (i.e. $\{t_3\}$) is distributed in the system, within the experiments we adopt the metric *job duration* since such metric measures the overall time spent by each GA to compute its own *global map*. Task t_1 is used by GAs to discover their own neighbourhood, a process which is also required by functions such as routing, sensor health monitoring, etc. already occurring within a WSN. The adjacency matrices gathered through t_1 could thus be obtained by simply piggybacking the messages employed by other functions performed in the system such as routing or health monitoring. Thus, we did not include the time budget t_1 in our measurements. In a fully connected network, each GA computes a *global map* by performing sequences of $\{t2, t4\}$, delegating the execution of $\{t_3\}$ to the TEs, until its *global map* contains the locations of all nodes belonging to the network. Notice that this procedure is carried out in parallel by all GAs in the network. Each sample measurement was obtained as the latency of all GAs in the WSN to complete their *global map* computation, averaged over approximatively 20 runs of the experiment. Although we did not intend to evaluate nor improve the accuracy of the CCA-MAP algorithm, we adopted the same set of input values (e.g. adjacency matrices, *local map* sizes) as those used in the previous simulations [Li and Kunz, 2009] and verified that sensors were indeed able to compute the positions with equal accuracy as reported by the simulator.

The case study evaluation was carried out by using the Auction algorithm (to distribute load through DWAG collaborations) with the BATS scheduling (to account for timely network conditions). The choice was made as result of the evaluations performed in Chapters 4 and 5. Moreover, compared to other heuristics, BATS is the simplest and the most robust in handling bandwidth changes. In the experiments, $weight_{CPU} \neq 0$ and $weight_{Band} \neq 0$ are used for BATS with bandwidth control, whilst $weight_{CPU} \neq 0$ and $weight_{Band} = 0$ are used for BATS without bandwidth control. Moreover, the values of the weights have been set to $weight_{CPU} = 16$ and $weight_{Band} = 1$, as for the experiments in Chapter 4.

Moreover, recall that for each experiment we report a statistical analysis conducted by applying One-Way Anova with Replication [Field and Hole, 2008] to the experimental values. The null hypothesis to be rejected each time is that the averages of all measured experimental sets are the same. Experimental results are determined to have a statistical impact when $p < .05$.

Let us finally specify that in the experimental results presented in Section 6.4.2 the *job duration* is averaged on the number of GAs involved.

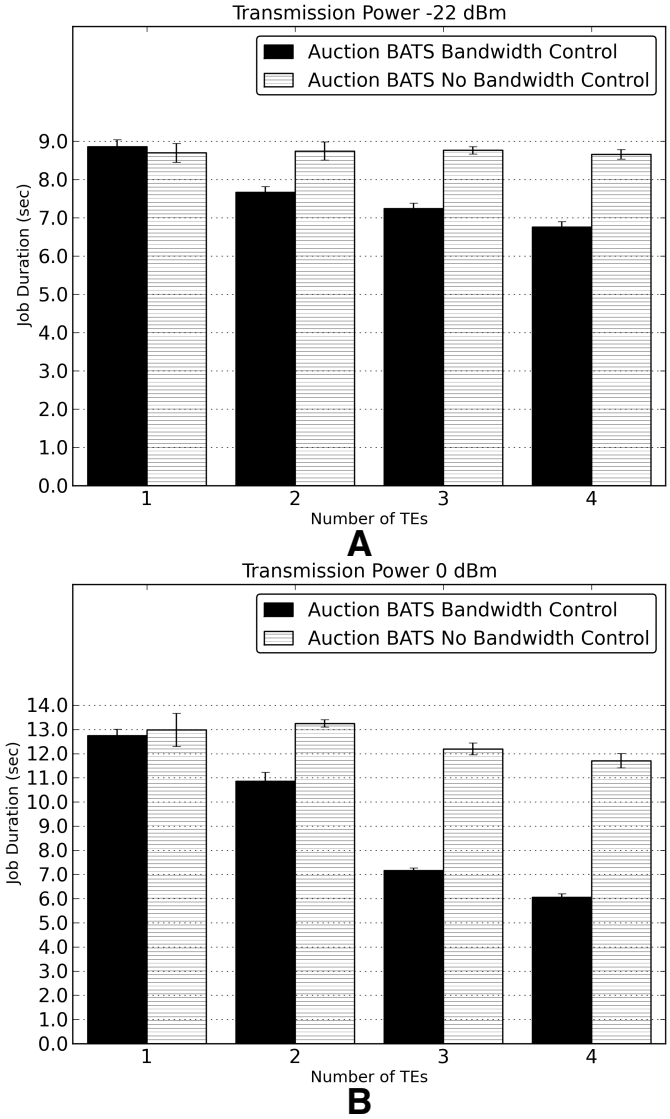


Figure 6.3: Effects of varying TE on job duration in network with 6 GAs with RF=-22 dBm (A) and 0 dBm (B).

Number TEs	F-value	p-value
1 TE	F(1,58)=4.8413	p=0.0318
2 TEs	F(1,58)=269.26	p=2.2e-16
3 TEs	F(1,58)=746.42	p=2.2e-16
4 TEs	F(1,58)=747.8	p=2.2e-16

Table 6.1: Anova analysis for job duration varying TE with RF=-22 dBm (Figure 6.3A).

Number TEs	F-value	p-value
1 TE	F(1,58)=2.154	p=0.1476
2 TEs	F(1,58)=148.70	p=2.2e-16
3 TEs	F(1,58)=655.83	p=2.2e-16
4 TEs	F(1,58)=582.13	p=2.2e-16

Table 6.2: Anova analysis for job duration varying TE with RF=0 dBm (Figure 6.3B).

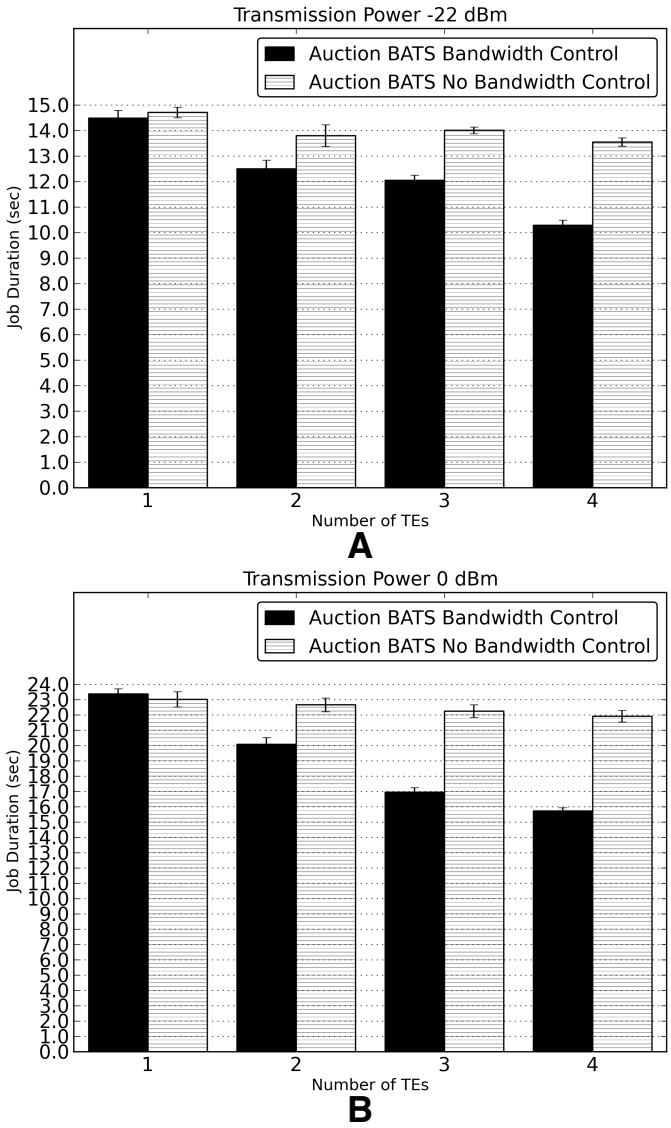


Figure 6.4: Effects of varying TE on job duration in network with 10 GAs with RF=-22 dBm (A) and 0 dBm (B).

Number TEs	F-value	p-value
1 TE	F(1,58)=2.503	p=0.1191
2 TEs	F(1,58)=16.117	p=0.0002
3 TEs	F(1,58)=86.857	p=3.953e-13
4 TEs	F(1,58)=114.64	p=2.324e-15

Table 6.3: Anova analysis for job duration varying TE with RF=-22 dBm (Figure 6.4A).

Number TEs	F-value	p-value
1 TE	F(1,58)=2.9467	p=0.0914
2 TEs	F(1,58)=61.232	p=1.204e-10
3 TEs	F(1,58)=261.83	p=2.2e-16
4 TEs	F(1,58)=379.43	p=2.2e-16

Table 6.4: Anova analysis for job duration varying TE with RF=0 dBm (Figure 6.4B).

6.4.2 Experimental Results

We are now in a position to describe the setting for each of the experiments and to reason about the collected results.

We conducted four sets of experiments, varying both the number and the radio transmission power of the GAs nodes. In particular, we used 6 GAs for the experiments illustrated in Figure 6.3 and 10 GAs for those in Figure 6.4. We set the radio transmission power of nodes to -22 dBm (the fourth lowest RF value in [Instruments, 2006]) and 0 dBm (the maximum RF value in [Instruments, 2006]) for the experiments in Figures 6.3-6.4A and Figures 6.3-6.4B, respectively. This latter parameter was varied to investigate the effects of generating both a loosely and a fully connected network: while with RF power -22 dBm not all nodes are in range of each other, with 0 dBm all nodes are in range. Thus, the variation of the radio transmission power affects both the size of the exchanged maps and consequently the number of radio transmissions to which every node is exposed. The radio transmission power of the TEs is set to 0 dBm, as for all the experiments conducted in Chapters 4 and 5.

Figure 6.3 and Figure 6.4 illustrate the measurements gathered. In particular, as stated in Section 6.4.1, we apply the Auction algorithm to distribute the load within the system and we integrate it with the BATS scheme. Hence we compare results achieved by exclusively accounting for computational capabilities of the auxiliary TE nodes, with those obtained by combining computation with local network conditions of the TEs responsible to handle collaborative job execution.

Effect of varying the number of the GAs: The aim of this experiment is to study the effect on the *job duration* brought by varying the number of GAs simultaneously required to perform CCA-MAP computations. We thus examine the *job duration* spent by the GAs to compute the *global map* of the GAs locations.

As illustrated in Figures 6.3 and 6.4, the *job duration* increases as follows: (i) when more GAs are involved in the CCA-MAP localisation process, because of the increased size of the *local maps* and the computed *global map*; and (ii) when the radio range grows bigger, because of the increased *local map* size and the increased radio interference. The overall latency in networks of 6 GAs with radio transmission power set to -22 dBm (Figure 6.3A) is, as a consequence, substantially lower than in networks of 10 GAs communicating with full radio power 0 dBm (Figure 6.4B). However, this behaviour does not imply that decreasing the radio transmission power of nodes is the best solution. In fact, as detailed in [Li and Kunz, 2009], a bigger map or a greater degree of connectivity (e.g. resulting from a more extended radio range) generate more accurate location results.

Effect of varying the number of the TEs: The aim of this experiment is to show the nature of the speed-up obtained if the number of the auxiliary TEs available for computation progressively increases. The experimental results, illustrated in both Figures 6.3 and 6.4, show that the increase in the number of TEs is effectively exploited only if the BATS scheme with bandwidth control is applied. On the other hand, if the GAs select the TEs exclusively based on the TEs' computational load, the increase of the candidate TEs does not lead to any significant performance improvement. In fact, the communication cost in offloading tasks to radio congested, although computationally capable TEs, often results in overall

job latencies longer than selecting congestion free TEs. Therefore, such findings confirm the behaviour reported in the experiments in both Chapters 4 and 5. In particular, when applying the BATS scheme with bandwidth control to the setting involving 6 GAs and 4 TEs, we observe performance improvements by $\sim$22% and $\sim$50%, as shown in Figure 6.3A-B respectively, when compared to the cases in which BATS exclusively accounts for the TEs computational load. Moreover, as illustrated in Figure 6.4A-B with the 10 GAs and 4 TEs setting, BATS with control of network information improves the performance by $\sim$28% and $\sim$27%, respectively.

The biggest improvement in performance of about 50% (Figure 6.3B), is detected when the network includes 6 GAs and 4 TEs, using the radio transmission power 0 dBm that covers all nodes. In this case, each GA has more candidate TEs from which to choose, thus it can avoid those that are congested. Moreover, if compared with larger networks of 10 GAs and 4 TEs where the bigger size of the exchanged maps considerably increases the computing latency, in the situation depicted in Figure 6.3 the communication latency makes up a relatively bigger proportion of the overall job execution latency. Thus, the saving on the communication latency by avoiding congested local area leads to bigger overall improvements. In addition, 4 TEs assisting 6 GA nodes would experience less overall computing load than serving 10 GA nodes. Note that *local maps* of 6 GA nodes was amongst the recommended configurations for tactical WSNs [Li and Kunz, 2009], hence the improvement brought up by BATS paradigm accounting for local network contention information can be of practical use.

The behaviours explained above are confirmed by the Anova results summarised in Tables 6.1 and 6.2 for the experiments with 6GAs (Figure 6.3) and Tables 6.3 and 6.4 for the experiments with 10 GAs (Figure 6.4).

Finally, the experimental results demonstrate that it is feasible to adopt the DWAG paradigm with the BATS scheduling scheme to implement distributed computing in a tactical WSNs, for even moderately complex algorithms. They also confirm that the BATS scheme can bring significant performance improvements to job execution latency.

6.5 Summary of Results

We now summarise the most interesting findings drawn from our experimental study, as follows:

- The experimental analysis shows the practical applicability of the DWAG paradigm with BATS heuristic in a real case study involving a collaborative localisation application representative of emergency scenarios.

- Considerable performance improvements are achieved whenever the BATS heuristic mechanism, combining the TEs computational and local network conditions, is integrated and applied to the load distribution algorithms. Hence experimental results show that faster completion times are achieved whenever informed decisions about heterogeneous traffic contention levels are accounted for in deciding towards which TE nodes to distribute load.

6.6 Discussion

In this chapter, we applied to an inherently distributed localisation case study, exclusively evaluated through simulations, our DWAG paradigm with bandwidth control. In particular, we described the motivation behind the selection of the localisation approach, proposed by Li et al. [Li and Kunz, 2009], as case study for our work. We briefly detailed the rationale behind the selected localisation scheme. We explained how we applied the DWAG paradigm with BATS scheme to the CCA-MAP localisation scheme. We thus matched the localisation algorithm requirements over the DWAG actors, and we described the additional algorithmic adaptations required to allow for the CCA-MAP approach to be deployed on testbeds. We then tested the DWAG paradigm with BATS heuristic applied to the CCA-MAP localisation scheme. Experimental results showed: (i) the practical applicability of the DWAG paradigm with BATS heuristic to a case study involving a localisation application representative for emergency scenarios; (ii) the strong benefits brought by accounting for local network information within BATS in performing job collaborations; (iii) conclusions in line with those drawn from artificially generated traffic.

Chapter 7

Conclusions and Future Work

The main goal of the work presented in this dissertation has been the empirical investigation of the impact of local network conditions on the distribution of computationally intensive applications that cannot be executed on single nodes, individually. More specifically, we presented the DWAG paradigm allowing computationally intensive applications to be collaboratively computed on resource-constrained devices. Then, we empirically investigated the effects of network traffic information on the system performance by distributing a range of applications. We devised and integrated within the adapted load sharing algorithms a set of heuristic offload mechanisms to measure the impacts of making informed offload decisions. Finally, we applied the DWAG paradigm with network control mechanism to a case study involving a localisation application.

The remainder of this last chapter is thus organised as follows: (i) in Section 7.1, we summarise the main contributions of this dissertation; (ii) in Section 7.2, we revise the system requirements; (iii) in Section 7.3, we provide a critical evaluation of our approach limitations; and (iv) in Section 7.4, we propose directions for future work.

7.1 Summary of Contributions

The aim of this dissertation has been the empirical analysis of the impact of local network conditions in distributing computationally intensive applications within deployed systems formed by resource-constrained devices. This has been achieved by designing algorithms and mechanisms both to distribute jobs and to simultaneously cope with local network traffic in order to achieve performance improvements in terms of job execution latency. The contributions of this dissertation are therefore organised according to the main sets of performed experimental evaluation and they are summarised as follows:

- **DWAG Paradigm:** Our work presented a Distributed Wireless Ad-hoc Grid paradigm devised to allow computationally intensive applications to achieve completion by exploiting the joint local capabilities of resource-constrained nodes collaborating with each other in an ad-hoc fashion to form virtual computational grids. We formalised the requirements for applications to allow DWAG applicability and we detailed DWAG main system flow, including the sequence of actions undertaken through DWAG execution. We then practically implemented DWAG on deployed WSNs testbeds.

- **Network Conditions Impact on Homogeneous Task Distribution:** Our work contributed the empirical investigation of the impact of network conditions on the performance of distributing homogeneous applications. More specifically, we proposed two load sharing algorithms, to distribute computationally intensive jobs. We implemented the algorithms on a deployed WSN testbed composed of TMote Sky devices. We devised and integrated within the load sharing algorithms a Bandwidth-Aware Task Scheduling offload mechanism that, dealing with both nodes computational capabilities and local network conditions, measured the actual effects of local network conditions on the overall job execution performance. In addition, we evaluated the effects of making decisions based on out-of-date information on the system performance.

- **Impact of Heterogeneous Applications:** Our work contributed the empirical investigation of the effects of combining network contention information with task profiling characterisation while distributing and collaboratively executing heterogeneous applications. We characterised several artificially generated application sets, profiled according to their computational and communication requirements. Thus, we generated heterogeneous job mixtures to be distributed. We evaluated the robustness of BATS mechanism against a P-BATS decision-making criteria combining task characterisation and network conditions. We devised a heuristic algorithm able to greedily compute an analytical lower-bound against which to compare the experimental data. Finally, we investigated the effects of making decisions based on out-of-date information, emphasising the consequent performance degradation.

- **Bandwidth-Aware DWAG to a Localisation Case Study:** Our work contributed the practical applicability of the devised DWAG paradigm with network control mechanism to a case study involving a collaborative localisation application. We described the localisation algorithm devised by Li et al. [Li and Kunz, 2009], we studied its limitations, and we proposed adaptations to allow DWAG applicability and deployment within testbeds. We integrated the load sharing algorithm with distribution mechanism with the localisation approach. In comparison to simulation, we analysed the real-world issues met by applying distributed algorithms on deployed systems. We evaluated the system performance and the improvements achieved by accounting for local network conditions during the decision-making phase.

7.2 Requirements Revision

In Chapter 2.1, we listed the requirements for the scenarios targeted within this dissertation. In this section, we evaluate the devised Bandwidth-Aware Distributed Wireless Ad-hoc Grids with respect to these criteria, as follows:

- **Distributed processing:** We devised a general DWAG paradigm independent from network structure, application definition, device deployment and physical characteristics. Thus, DWAG is a lightweight, totally distributed paradigm in which resource-constrained nodes exploit their social capabilities by autonomously synchronising with each other forming virtual dynamic ad-hoc grids

aiming at collaboratively performing computationally intensive computations. This was done with the aim of improving task completion time in performing timely computations.

This idea contrasts therefore with most of the approaches relying, instead, upon the capabilities of subsets of more powerful nodes hierarchically organised and in charge to perform the most complex and demanding computations.

- **Network communications:** The DWAG paradigm explicitly accounts for the impact of heterogeneous environmental network conditions while distributing computational load among nodes. This is done through the definition of heuristics fetching radio information from the physical layer, thus exploiting a cross-layer design of the approach, and integrating it directly within the algorithms in order to measure its impacts on distribution performance.

  We thus practically explore the tradeoff between the distribution of computation, needed to enhance the computational abilities of networks of small nodes, and the creation of network traffic that results from that distribution. The highly deployment-specific nature of radio communications means that simulations, that are capable of producing validated, high-quality results, are extremely hard to construct. Consequently, to produce credible results, our experiments used a direct empirical analysis based on a deployed network of devices located at UCL. The results emerging from our evaluation indeed show the benefits gained in accounting for network conditions in performing collaborative computations.

- **Implementation in deployed systems:** The distributed Bandwidth-Aware Distributed Wireless Ad-hoc Grids approach has been implemented on a deployed testbed of devices. The choice was made with the idea of measuring the actual impact of network conditions on the performance of distributed computations and thus balancing both computational and communication load.

  Experimental results have been collected within deployed systems running a variety of experimental configurations and different CPU-bound, I/O-bound and mixed tasks settings. In addition, we have taken a realistic application, based on location estimation, and implemented that across the same network with results that support the conclusions drawn from the artificially generated traffic. Using this setup, we have established that even relatively simple load sharing algorithms are capable of achieving considerable performance improvements over a range of different artificially generated scenarios with timely contextual information. The DWAG paradigm has thus been proved to work in practice and to be lightweight enough to run and be supported by resource-constrained devices.

7.3 Critical Evaluation

The work presented in this dissertation is mainly focused on the pragmatical and empirical analysis of the impact of local network conditions in distributing computationally intensive applications within deployed systems formed by resource-constrained devices. The current work limitations are, therefore, mainly related to some of the experimental assumptions, made during the evaluation, and to some pa-

rameters representing sources of variability for the experimental results. The limitations are summarised as follows:

- *Assumption about constant radio connectivity over individual task offload:* In the experimental evaluation, we assumed that system nodes are moving sufficiently slowly so that radio connections are unlikely to break before offloaded tasks reach completion, and thus computed results are uploaded back from the TEs to the GAs. The assumption is reasonable for the targeted scenarios in which involved applications are almost real-time computed (e.g. in emergency scenarios, information needs to be promptly on-the-field processed since the first minutes after a threat are the most crucial ones). However, whenever offloaded tasks require considerable amount of time to be computed (e.g. several hours of computation), the GAs might no longer be in reach of the TEs and thus they might not be able to collect results derived from offloaded task computation. Additional mechanisms should be devised and introduced within the algorithms to deal with such situations, and they could be responsible for delaying the job execution.

- *Assumption about task execution priority:* In the experiments, every task into which a job is split has the same execution priority, thus task execution cannot be interrupted because of the presence of high-priority tasks. However, whenever tasks to be executed have higher priority than others, such high-priority tasks should be able to preempt task execution. Additional mechanisms should be added within the algorithms to deal with different task priorities. Concurrency combined with task priorities could thus be responsible for favouring or delaying job execution according to the execution priority.

- *Assumption about sink nodes disregarded in the experimental evaluation:* In the experimental evaluation, we do not explicitly introduce sink nodes and routing algorithms to disseminate to external authorities the information once it is locally computed. In fact, our research hypothesis focuses on locally analysing the impact of network conditions whenever node collaborations become necessary to achieve job completion. Our approach aims at distributing not only computational but also communication load in such a way that areas avoid to become over-congested. The interferer node, introduced in the experimental evaluation to reproduce heterogeneous areas of network contention, emulates the presence within the system of sink nodes traversed by considerable traffic. In the future, it might be interesting to introduce actual sink nodes and additional dissemination protocols in the system to be able to compute the whole time spent from information computation and actual delivery to external authorities.

- *Assumption about job characterisations:* In the experiments, we chose significant task computational and communication configurations mirroring settings deriving from the computational grids field and tuning the parameters to work within WSNs scenarios. Although in the current dissertation we have explored an experimental spectrum and we have reported the most significant results, the overall analysis does not claim to be exhaustive. The behaviour of more complex distributions could be explored and compared against those adopted (i.e. Gaussian and Poissonian). Moreover,

we have applied our approach to a case study involving a localisation application because (i) localisation is of primary concern within emergency scenarios, and (ii) the valuable opportunity to collaborate with Communications Research Centre, Canada helping researchers to port in practice a theoretical approach so far exclusively evaluated through simulations. Since implementations on deployed systems are considerably demanding and time consuming, we confined the scope of this dissertation to node localisation. However, it would be interesting to apply in future the devised approach to a multitude of case studies and feed the system with values representative of WSN applications.

- *Deployment Heterogeneity:* In the experiments, we adopted the number of running processes to represent the computational capability of a node, since we dealt with resource-constrained devices. In scenarios with heterogeneous more powerful devices, other metrics might be more adequate to represent the CPU usage of a node. In this perspective, our approach might, therefore, be required to integrate an additional layer providing at the application layer a common vision of the information fetched at the physical layer, in a transparent way from the device adopted. Similar considerations would apply to the network traffic and task profile characterisations.

- *Parameter tuning:* In the experimental evaluation, parameters were tuned according to values representative of computation and network traffic for the adopted deployment. Different deployments would, thus, similarly require a fast bootstrap phase to allow parameters to tune their values (e.g. the weight values within BATS) according to the network traffic and nodes' computation capabilities.

7.4 Future Work

In the future, we would like to lift some of the limitations discussed in Section 7.3.

Firstly, we would like to make the algorithms robust against radio connectivity failures. More specifically, we would like to expand them to have additional control mechanisms so that, whenever nodes are moving sufficiently fast that radio connections are likely to break before offloaded tasks reach completion, it could still be possible to exploit multi-hop radio connectivity among nodes to disseminate the computed information without repeating the offload procedure.

Secondly, we would like to apply the DWAG approach with BATS, accounting for network information, to a multitude of case studies in order to compare the actual benefits brought by the paradigm within each of them against the localisation application.

Thirdly, we would like to introduce additional mechanisms within the algorithm to deal with heterogeneous task execution priorities.

Fourthly, we would like to introduce actual sink nodes in the system, and thus dissemination and routing algorithms, in order to evaluate the whole time spent from application computation to result delivery to external authorities.

In the current dissertation, we did not focus on devising novel sophisticated load sharing algorithms since the focus of our research hypothesis relied upon analysing the local impact of network conditions

on distribution whenever node collaborations were necessary to achieve job completion. Moreover, our experimental results also proved how, even relatively simple load sharing algorithms, were capable of achieving considerable performance improvements over a range of different experimental settings and configurations. In fact, no additional improvements were detected by increasing the level of sophistication of the single algorithms. However, we would like to exploit the presence of algorithms already in place in the environment for different purposes (e.g. routing, security, dissemination, etc.) to ameliorate the distribution of both computational and communication load and, at the same time, to deal with the risk of out-of-date information.

Moreover, we would like to test the DWAG paradigm with BATS, accounting for network information, directly on real-world fields, thus collecting results not only from office-like indoor environments and configurations, but also from actual emergency-like kind of scenarios. In fact, especially during emergencies, the amount of in-place communication becomes a crucial aspect to be tackled to avoid the isolation of entire areas caused by massive congestive collapse. In this context, we would like to test the approach at the presence of real node movements. For these purposes, we initiated a collaboration with Professor Galea and its team at University of Greenwich, UK.

Finally, we would like to integrate within our approach additional security mechanisms to improve system robustness and guarantee data integrity against spoofing and man-in-the-middle attacks.

In conclusion, in this dissertation, we proposed a pragmatic and empirical analysis of the impact of local network conditions in distributing computationally intensive applications within deployed systems formed by resource-constrained devices. Adopting evaluations on deployed testbeds, we devised the DWAG paradigm, we implemented concurrent load sharing algorithms and mechanisms to balance both computational and communication system load accounting for timely local network conditions. We tested the devised techniques across a set of artificially generated task mixtures and a realistic case study based on location estimation. However, there are still experimental assumption that could be lifted to allow additional evaluation and possible algorithm modification with the aim to ameliorate system performance.

Bibliography

[AASK, 2010] AASK (2010). http://fseg2.gre.ac.uk/AASK/.

[Abbasi and Younis, 2007] Abbasi, A. A. and Younis, M. (2007). A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14-15):2826–2841.

[Abrach et al., 2003] Abrach, H., Bhatti, S., Carlson, J., Dai, H., Rose, J., Sheth, A., Shucker, B., and Han, R. (2003). MANTIS: system support for MultimodAl NeTworks of In-situ Sensors. In *Proceedings of the 2^{nd} annual ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2003)*, pages 50–59.

[Abrams et al., 2006] Abrams, Z., Chen, H.-L., Guibas, L., Liu, J., and Zhao, F. (2006). Kinetically Stable Task Assignment for Networks of Microservers. In *Proceedings of the 5^{th} annual ACM/IEEE International Conference on Information Processing In Sensor Networks (IPSN 2006)*, pages 93–101.

[Africa@Home, 2010] Africa@Home (2010). http://africa-at-home.web.cern.ch/africa%2Dat%2Dhome/.

[Aguayo et al., 2004] Aguayo, D., Bricket, J., Biswas, S., Judd, G., and Morris, R. (2004). Link-level Measurements from an 802.11b Mesh Network. In *Proceedings of the 2004 ACM Conference of the Special Interest Group on Data Communication (SIGCOMM 2004)*, pages 121–132.

[Ahuja and Myers, 2006] Ahuja, S. P. and Myers, J. R. (2006). A Survey on Wireless Grid Computing. *The Journal of Supercomputing*, 37(1):3–21.

[Akkaya and Younis, 2005] Akkaya, K. and Younis, M. (2005). A survey of routing protocols in wireless sensor networks. *Elsevier Ad Hoc Network Journal*, 3(3):325–349.

[Al-Ars and Kootkar, 2007] Al-Ars, Z. and Kootkar, S. (2007). Design and Implementation of Reliable Wireless Sensor Networks—A Case Study in Commuter Trains. In *Proceedings of the 2007 Workshop of Program for Research on Integrated Systems and Circuits (ProRISC 2007)*, pages 303–306.

[Ali et al., 2008] Ali, M., Böhm, A., and Jonsson, M. (2008). Wireless Sensor Networks for Surveillance Applications - A Comparative Survey of MAC Protocols. In *Proceedings of the 4^{nd} International Conference on Wireless and Mobile Communications (ICWMC 2008)*, pages 399–403.

[Ali et al., 2006] Ali, M., Saif, U., Dunkels, A., Voigt, T., Römer, K., Langendoen, K., Polastre, J., and Uzmi, Z. A. (2006). Medium Access Control Issues in Sensor Networks. *ACM SIGCOMM Computer Communication Review*, 36(2):33–36.

[Amundson and Koutsoukos, 2009] Amundson, I. and Koutsoukos, X. D. (2009). A Survey on Localization for Mobile Wireless Sensor Networks. In *Proceedings of the* 2^{nd} *International Workshop on Mobile Entity Localization and Tracking in GPS-less Environments (MELT 2009)*, pages 235–254.

[Antonis et al., 2004] Antonis, K., Garofalakis, J., Mourtos, I., and Spirakis, P. (2004). A hierarchical adaptive distributed algorithm for load balancing. *Journal of Parallel and Distributed Computing*, 64(1):151–162.

[Arampatzis et al., 2005] Arampatzis, T., Lygeros, J., and Manesis, S. (2005). A Survey of Applications of Wireless Sensors and Wireless Sensor Networks. In *Proceedings of the* 13^{th} *annual IEEE International Symposium on Mediterranean Conference on Control and Automation*, pages 719–724.

[Arduino, 2010] Arduino (2010). `http://www.arduino.cc/`.

[Bacco et al., 2004] Bacco, G. D., Melodia, T., and Cuomo, F. (2004). A MAC Protocol for Delay-Bounded Applications in Wireless Sensor Networks. In *Proceedings of the* 3^{rd} *annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2004)*.

[Baronti et al., 2007] Baronti, P., Pillai, P., Chook, V. W., Chessa, S., Gotta, A., and Fun, Y. (2007). Wireless sensor networks: A survey on the state of the art and the 802.15.4 and Zigbee standards. *Computer Communications*, 30(7):1655–1695.

[Basaran et al., 2006] Basaran, C., Baydere, S., Bongiovanni, G., Dunkels, A., Ergin, M. O., Feeney, L. M., Hacioglu, I., Handziski, V., Kø"pke, A., Lijding, M., Gaia Maselli and, N. M., Petrioli, C., Santini, S., van Hoesel, L., Voigt, T., and Zanella, A. (2006). Research Integration: Platform Survey Critical Evaluation of Platforms Commonly Used in Embedded Wisents Research. Technical Report Technical Report EW-T21/D01-SICS-001-01.

[Bertocco et al., 2008] Bertocco, M., Gamba, G., and Sona, A. (2008). Assessment of Out-of-Channel Interference Effects on IEEE 802.15.4 Wireless Sensor Networks. In *Proceedings of the 2008 annual IEEE Instrumentation and Measurement Technology Conference (IMTC 2008)*, pages 1712–1717.

[Bhatti and Xu, 2009] Bhatti, S. and Xu, J. (2009). Survey of Target Tracking Protocols Using Wireless Sensor Network. In *Proceedings of the* 5^{th} *annual International Conference on Wireless and Mobile Communications (ICWMC 2009)*, pages 110–115.

[Burrell et al., 2004] Burrell, J., Brooke, T., and Beckwith, R. (2004). Vineyard computing: sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1):38–45.

[Buyya, 1999a] Buyya, R. (1999a). *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall.

[Buyya, 1999b] Buyya, R. (1999b). *High Performance Cluster Computing: Programming and Applications*. Prentice Hall.

[Campbell et al., 2006] Campbell, A. T., Eisenman, S. B., Lane, N. D., Miluzzo, E., and Peterson, R. A. (2006). People-Centric Urban Sensing. In *Proceedings of the* 2^{nd} *annual ACM International Workshop on Wireless Internet (WICON 2006)*, page 18.

[Cerpa et al., 2003] Cerpa, A., Busek, N., and Estrin, D. (2003). SCALE: A tool for Simple Connectivity Assessment in Lossy Environments. Technical Report Technical Report CENS-21, UCLA.

[Cha et al., 2007] Cha, H., Choi, S., Jung, I., Kim, H., Shin, H., Yoo, J., and Yoon, C. (2007). RETOS: Resilient, expandable, and threaded operating system for wireless sensor networks. In *Proceedings of the* 6^{th} *annual ACM International Conference on Information Processing in Sensor Networks (IPSN 2007)*, pages 148–157.

[Chiasserini and Rao, 2002] Chiasserini, C. F. and Rao, R. R. (2002). On the Concept of Distributed Digital Signal Processing in Wireless Sensor Networks. In *Proceedings of the* 21^{st} *annual IEEE/Boeing Military Communication Conference (MILCOM 2002)*, pages 260–264.

[Chuang and Cheng, 2002] Chuang, P.-J. and Cheng, C.-W. (2002). On File and Task Placements and Dynamic Load Balancing in Distributed Systems. *Tamkang Journal of Science and Software Engineering*, 5(4):241–252.

[CodeBlue, 2010] CodeBlue (2010). `http://www.codeblue.com/`.

[Colvin, 1983] Colvin, A. (1983). CSMA with collision avoidance. *Computer Communications*, 6(5):227–235.

[Costa et al., 2007] Costa, P., Coulson, G., Gold, R., Lad, M., Mascolo, C., Mottola, L., Picco, G. P., Sivaharan, T., Weerasinghe, N., and Zachariadis, S. (2007). The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario. In *Proceedings of the* 5^{th} *annual International Conference on Pervasive Communications (PerCom 2007)*, pages 69–78.

[Coulson, 2006] Coulson, G. (2006). Pervasive Grids: Integrating Sensor Networks Into the Fixed Grid. In *Proceedings of the* 2^{nd} *annual IEEE/ACM Euro-American Workshop on Middleware for Sensor Networks (invited paper), co-located with the International Conference on Distributed Computing in Sensor Systems (DCOSS 2006)*.

[Coulson et al., 2006] Coulson, G., Kuo, D., and Brooke, J. (2006). Sensor Networks + Grid Computing = A New Challenge for the Grid? *IEEE Distributed Systems Online*, 7(12).

[Crossbow, 2007] Crossbow (2007). Avoiding RF Interference Between WiFi and Zigbee. Technical Report Technical Report - Crossbow.

[de Freitas et al., 2009] de Freitas, E. P., Heimfarth, T., Pereira, C. E., Ferreira, A. M., Wagner, F. R., and Larsson, T. (2009). Multi-Agent Support in a Middleware for Mission-Driven Heterogeneous Sensor Networks. *The Computer Journal*.

[Demartines and Herault, 1997] Demartines, P. and Herault, J. (1997). Curvilinear Component Analysis: A Self-Organizing Neural Network for Nonlinear Mapping of Data Sets. *IEEE Transactions on Neural Networks*, 8(1):148–154.

[Demirkol et al., 2006] Demirkol, I., Ersoy, C., and Alagöz, F. (2006). MAC Protocols for Wireless Sensor Networks: a Survey. *Communications Magazine, IEEE Press*, 44(4):115–121.

[Desch, 2001] Desch, M. C. (2001). Soldiers in Cities: Military Operations on Urban Terrain. Technical Report ISBN 1-58487-062-1.

[Deshpande et al., 2005] Deshpande, A., Guestrin, C., and Madden, S. R. (2005). Resource-Aware Wireless Sensor-Actuator Networks. *IEEE Data Engineering*, 28(1).

[Dong and Akl, 2006] Dong, F. and Akl, S. G. (2006). Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical Report Technical Report - 2006-504.

[Drineas et al., 2006] Drineas, P., Magdon-Ismail, M., Pandurangan, G., Virrankoski, R., and Savvides, A. (2006). Distance Matrix Reconstruction from Incomplete Distance Information for Sensor Network Localization. In *Proceedings of the* 3^{rd} *annual IEEE Communications Society Conference on Sensor, Mesh, and Ad Hoc Communications and Networks (SECON 2006)*, pages 536–544.

[Dunkels, 2003] Dunkels, A. (2003). Full TCP/IP for 8-bit architectures. In *Proceedings of the* 1^{st} *International Conference on Mobile Systems, Applications, and Services (MobiSys 2003)*, pages 85–98.

[Dunkels et al., 2004] Dunkels, A., Grönvall, B., and Voigt, T. (2004). Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the* 1^{st} *annual ACM International Workshop on Embedded Networked Sensors (SenSys 2004)*, pages 455–462.

[Dunkels et al., 2007] Dunkels, A., Österlind, F., and He, Z. (2007). An Adaptive Communication Architecture for Wireless Sensor Networks. In *Proceedings of the* 5^{th} *annual ACM International Conference on Embedded Networked Sensor Systems (SenSys 2007)*, pages 335–349.

[Dunkels and Schmidt, 2005] Dunkels, A. and Schmidt, O. (2005). Protothreads - Lightweight Stackless Threads in C. Technical Report Technical Report T2005-05, SICS - Swedish Institute of Computer Science.

[Dunkels et al., 2005] Dunkels, A., Schmidt, O., and Voigt, T. (2005). Using Protothreads for Sensor Node Programming. In *Proceedings of the* 1^{st} *annual ACM International Workshop on Real-World Wireless Sensor Networks (REALWSN 2005)*.

[Dunkels et al., 2006] Dunkels, A., Schmidt, O., Voigt, T., and Ali, M. (2006). Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. In *Proceedings of the* 4^{th} *annual ACM International Conference on Embedded Networked Sensor Systems (SenSys 2006)*, pages 29–42.

[Eager et al., 1986] Eager, D. L., Lazowska, E. D., and Zahorjan, J. (1986). Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering*, 12(5):662–675.

[EasySen, 2010] EasySen (2010). `www.easysen.com`.

[Emmerich et al., 2005] Emmerich, W., Butchart, B., Chen, L., Wassermann, B., and Price, S. L. (2005). Grid Service Orchestration Using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3(3-4):283–304.

[Ergen et al., 2009] Ergen, E., Sariel-Talay, S., and Guven, G. (2009). Providing Local Information for Search and Rescue Using Sensor-Based Local Databases. In *Proceedings of the 2009 International Workshop on Computing in Civil Engineering (ASCE 2009)*, pages 43–52.

[Eriksson et al., 2007] Eriksson, J., Dunkels, A., Finne, N., Österlind, F., and Voigt, T. (2007). MSPsim - An Extensible Simulator for MSP430-Equipped Sensor Boards. In *Proceedings of the* st *annual European Conference on Wireless Sensor Networks (EWSN 2007)*.

[Erman et al., 2008] Erman, A., Hoesel, L., and Havinga, P. (2008). Enabling mobility in heterogeneous wireless sensor networks cooperating with UAVs for mission-critical management. *IEEE Wireless Communications*, 15(6):38–46.

[Eswaran et al., 2005] Eswaran, A., Rowe, A., and Rajkumar, R. (2005). Nano-RK: an Energy-aware Resource-centric RTOS for Sensor Networks. In *Proceedings of the* 26^{th} *annual IEEE International Symposium on Real-Time Systems (RTSS 2005)*, pages 256–265.

[EXODUS, 2010] EXODUS (2010). `http://fseg.gre.ac.uk/exodus/index.html`.

[Fasolo et al., 2007] Fasolo, E., Rossi, M., Widmer, J., and Zorzi, M. (2007). In-network aggregation techniques for wireless sensor networks: a survey. *Wireless Communications, IEEE*, 14(2):70–87.

[Ferguson et al., 1988] Ferguson, D., Yemini, Y., and Nikolaou, C. (1988). Microeconomic Algorithms for Load Balancing in Distributed systems. In *Proceedings of the* 8^{th} *annual IEEE International Conference on Distributed Computer Systems (ICDCS 1988)*, pages 491–499.

[Ferranti et al., 2008] Ferranti, E., Trigoni, N., and Levene, M. (2008). Robot-Assisted Discovery of Evacuation Routes in Emergency Scenarios. In *Proceedings of the* 17^{th} *annual IEEE International Conference on Robotics and Automation (ICRA 2008)*, pages 2824–2830.

[Ferranti et al., 2009] Ferranti, E., Trigoni, N., and Levene, M. (2009). Rapid Exploration of Unknown Areas Through Dynamic Deployment of Mobile and Stationary Sensor Nodes. *Journal of Autonomous Agents and Multi-Agent Systems, Springer Press*, 19(2):210–243.

[Field and Hole, 2008] Field, A. and Hole, G. J. (2008). *How to Design and Report Experiments*. SAGE.

[Fok et al., 2005] Fok, C., C.Roman, G., and Lu, C. (2005). Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. In *Proceedings of the* 24^{th} *annual International Conference on Distributed Computing Systems*, pages 653–662.

[Folding@Home, 2010] Folding@Home (2010). `http://folding.stanford.edu/`.

[Foster and Kesselman, 1997] Foster, I. and Kesselman, C. (1997). Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128.

[Foster and Kesselman, 1999] Foster, I. and Kesselman, C. (1999). *The Grid - Blueprint for a new computing infrastructure*. Morgan Kaufmann.

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3):220–222.

[Fuggetta et al., 1998] Fuggetta, A., Picco, G. P., and Vigna, G. (1998). Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361.

[Galea et al., 1993] Galea, E. R., Galparsoro, J. M. P., and Pearce, J. (1993). A Brief Description of the EXODUS Evacuation Model. In *Proceedings of the* 18^{th} *annual International Conference on Fire Safety*, pages 149–162.

[Ganesan et al., 2002] Ganesan, D., Krishnamachari, B., Woo, A., Culler, D., Estrin, D., and Wicker, S. (2002). Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Technical Report Technical Report CSD-TR 02-0013, UCLA.

[Garey and Johnson, 1979] Garey, M. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company.

[Gaynor et al., 2004] Gaynor, M., Moulton, S., Welsh, M., LaCombe, E., Rowan, A., and Wynne, J. (2004). Integrating Wireless Sensor Networks with the Grid. *Internet Computing, IEEE Press*, 8(4):32–39.

[Giannecchini et al., 2004] Giannecchini, S., Caccamo, M., and Shih, C.-S. (2004). Collaborative Resource Allocation in Wireless Sensor Networks. In *Proceedings of the* 16^{th} *annual IEEE Euromicro Conference on Real-Time Systems (ECRTS 2004)*, pages 35–44.

[Gil-Castieira et al., 2008] Gil-Castieira, F., Gonzlez-Castao, F., Duro, R. J., and Lopez-Pea, F. (2008). Urban Pollution Monitoring through Opportunistic Mobile Sensor Networks Based on Public Transport. In *Proceedings of the* 6^{th} *annual IEEE International Conference on Computational Intelligence for Measurement Systems And Applications (CIMSA 2008)*, pages 70–74.

[gLite, 2010] gLite (2010). `http://glite.web.cern.ch/glite/`.

[Gmach et al., 2009] Gmach, D., Rolia, J., Cherkasova, L., and Kemper, A. (2009). Resource pool management: Reactive versus proactive or lets be friends. *Computer Networks*, 53(17):2905–2922.

[Grama et al., 2003] Grama, A., Gupta, A., Karypis, G., and Kumar, V. (2003). *Introduction to Parallel Computing*. Pearson - Addison Wesley.

[Gumstix, 2010] Gumstix (2010). `http://www.gumstix.com/`.

[Han et al., 2005] Han, C.-C., Kumar, R., Shea, R., Kohler, E., and Srivastava, M. (2005). A Dynamic Operating System for Sensor Nodes. In *Proceedings of the* 3^{rd} *annual ACM International Conference on Mobile systems, Applications, and Services (MobiSys 2005)*, pages 163–176.

[Heinzelman et al., 2002] Heinzelman, W. B., Chandrakasan, A. P., and Balakrishnan, H. (2002). An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670.

[Heinzelman et al., 2004] Heinzelman, W. B., Murphy, A. L., Carvalho, H. S., and Perillo, M. A. (2004). MiLAN: Middleware linking applications and networks. *IEEE Network Magazine Special Issue*, 18.

[HEN, 2010] HEN (2010). Heterogeneous Experimental Network (HEN) - University College London. `http://www.cs.ucl.ac.uk/research/hen/`.

[Hill et al., 2000] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K. (2000). System Architecture Directions for Networked Sensors. In *Proceedings of the* 9^{th} *annual IEEE/ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, pages 93–104.

[Hughes et al., 2006] Hughes, D., Greenwood, P., Coulson, G., and Blair, G. (2006). GridStix: Supporting Flood Prediction using Embedded Hardware and Next Generation Grid Middleware. In *Proceedings of the* 4^{th} *annual IEEE International Workshop on Mobile Distributed Computing, co-located with the International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2006)*, pages 621–626.

[Imote2, 2010] Imote2 (2010). `http://www.xbow.com/Products/productdetails.aspx?sid=253`.

[Instruments, 2006] Instruments, T. (2006). 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. Technical Report Chipcon Products from Texas Instruments - SWR6041.

[Intanagowiwat et al., 2000] Intanagowiwat, C., Govindan, R., and Estrin, D. (2000). Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the* 6^{th} *annual IEEE/ACM International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 56–67.

[Izakian et al., 2010] Izakian, H., Abraham, A., and Ladani, B. T. (2010). An auction method for resource allocation in computational grids. *Future Generation Computer Systems, Elsevier*, 26(2):228–235.

[Jayasuriya et al., 2004] Jayasuriya, A., Perreau, S., Dadej, A., and Gordon, S. (2004). Hidden vs. exposed terminal problem in ad hoc networks. In *Proceedings of the Australian Telecommunication Networks and Applications Conference (ATNAC 2004)*.

[Juang et al., 2002] Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L.-S., and Rubenstein, D. (2002). Energy-Efficient Computing For Wildlife Tracking: Design Trade-offs And Early Experiences With ZebraNet. In *Proceedings of the* 10^{th} *annual International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X 2002)*, pages 96–107.

[Karl and Willig, 2005] Karl, H. and Willig, A. (2005). *Protocols and Architectures for Wireless Sensor Networks*. John Wiley and Sons.

[Kasten and Römer, 2005] Kasten, O. and Römer, K. (2005). Beyond event handlers: Programming wireless sensors with attributed state machines. In *Proceedings of the* 4^{th} *annual Symposium on Information Processing in Sensor Networks (IPSN 2005)*, pages 45–52.

[Kho et al., 2007] Kho, J., Rogers, A., and Jennings, N. R. (2007). Decentralised Adaptive Sampling of Wireless Sensor Networks. In *Proceedings of the* 1^{st} *annual International Workshop on Agent Technology for Sensor Networks (ATSN 2007)*, pages 14–18.

[Kleinrock and Tobagi, 1975] Kleinrock, L. and Tobagi, F. (1975). Packet Switching in Radio Channels Part II - the Hidden Node Problem in Carrier Sense Multiple Access Nodes and the Busy Tone Solution. *IEEE Transactions on Communications*, 23(12):1417–1433.

[Kotz et al., 2003] Kotz, D., Newport, C., and Elliot, C. (2003). The mistaken axioms of wireless-network research. Technical Report Technical Report TR2003-467, Dartmouth College Computer Science.

[Kotz et al., 2004] Kotz, D., Newport, C., Gray, R. S., Liu, J., Yuan, Y., and Elliott, C. (2004). Experimental Evaluation of Wireless Simulation Assumptions. Technical Report Technical Report TR2004-507, Dartmouth College Computer Science.

[Kulik et al., 1999] Kulik, J., Rabiner, W., and Balakrishnan, H. (1999). Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proceedings of the* 5^{th} *annual IEEE/ACM International Conference on Mobile Computing and Networking (MobiCom 1999)*, pages 174–185.

[Kuorilehto et al., 2005] Kuorilehto, M., Hännikäinen, M., and Hämäläinen, T. D. (2005). A Survey of Application Distribution in Wireless Sensor Networks. *EURASIP Journal on Wireless Communications and Networking, ACM*, 5(5):774–788.

[l. F. Akyildiz and Kasimoglu, 2004] l. F. Akyildiz and Kasimoglu, I. H. (2004). Wireless sensor and actor networks: Research challenges. *Journal of Ad Hoc Networks, Elsevier Press*, 2(4):351–367.

[l. F. Akyildiz et al., 2002] l. F. Akyildiz, Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless Sensor Networks: a Survey. *Computer Networks, Elsevier Press*, 38(4):393–422.

[Lau et al., 2006] Lau, S.-M., Lu, Q., and Leung, K.-S. (2006). Adaptive Load Distribution Algorithms for Heterogeneous Distributed Systems with Multiple Task Classes. *Journal of Parallel and Distributed Computing, ELSEVIER Press*, 66(2):163–180.

[LEAD, 2010] LEAD (2010). `https://portal.leadproject.org/gridsphere/gridsphere`.

[Lei et al., 2009] Lei, C., Szymanski, B. K., and Branch, J. W. (2009). Auction-Based Congestion Management for Target Tracking in Wireless Sensor Networks. In *Proceedings of the* 7^{th} *annual IEEE International Conference on Pervasive Computing and Communications (PERCOM 2009)*.

[Leung et al., 2008] Leung, H., Chandana, S., and Wei, S. (2008). Distributed sensing based on intelligent sensor networks. *IEEE Circuits and Systems Magazine*, 8(2):38–52.

[Levis and Culler, 2002] Levis, P. and Culler, D. (2002). Mate: A tiny virtual machine for sensor networks. In *Proceedings of the* 10^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems*.

[Levis et al., 2003] Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the* 1^{st} *annual ACM International Conference on Embedded Networked Sensor Systems (SenSys 2003)*, pages 126–137.

[LHC@Home, 2010] LHC@Home (2010). `http://lhcathome.cern.ch/`.

[Li, 2008] Li, L. (2008). Localization in Self-Healing Autonomous Sensor Networks (SASNet). Technical Report Technical Report TR 2008-020, DRDC Ottawa.

[Li and Kunz, 2007a] Li, L. and Kunz, T. (2007a). Cooperative Node Localization for Tactical Wireless Sensor Networks. In *Proceedings of the* 26^{th} *annual IEEE/Boeing Military Communication Conference (MILCOM 2007)*, pages 1–7.

[Li and Kunz, 2007b] Li, L. and Kunz, T. (2007b). Localization Applying An Efficient Neural Network Mapping. In *Proceedings of the* 1^{st} *ACM International Conference on Autonomic Computing and Communication Systems (Autonomics 2007)*, pages 1–9.

[Li and Kunz, 2009] Li, L. and Kunz, T. (2009). Cooperative Node Localization Using Non-Linear Data Projection. *ACM Transactions on Sensor Networks (TOSN)*, 5(1):1–26.

[Li et al., 2004] Li, S., Lin, Y., Son, S., Stankovic, J., and Wei, Y. (2004). Event detection services using data service middleware in distributed sensor networks. *Telecommunication Systems*, 26(2).

[Lim et al., 2007] Lim, H. B., Ling, K. V., Wang, W., Yao, Y., Iqbal, M., Li, B., Yin, X., and Sharma, T. (2007). The National Weather Sensor Grid. In *Proceedings of the* 5^{th} *annual ACM International Conference on Embedded Networked Sensor Systems (SenSys 2007)*, pages 369–370.

[Lim et al., 2005] Lim, H. B., Teo, Y. M., Mukherjee, P., Lam, V. T., Wong, W. F., and See, S. (2005). Sensor Grid: Integration of Wireless Sensor Networks and the Grid. In *Proceedings of the* 30^{th} *annual IEEE International Conference on Local Computer Networks (LCN 2005)*, pages 91–98.

[Lindsey et al., 2002] Lindsey, S., Raghavendra, C., and Sivalingam, K. M. (2002). Data Gathering Algorithms in Sensor Networks using Energy Metrics. *IEEE Transaction on Parallel and Distributed Systems*, 13(9):924–935.

[Liu et al., 2006] Liu, X., Wang, Q., He, W., Caccamo, M., and Sha, L. (2006). Optimal Real-Time Sampling Rate Assignment for Wireless Sensor Networks. *ACM Transactions on Sensor Networks, ACM Press*, 2(2):263–295.

[Lorincz et al., 2004] Lorincz, K., Malan, D. J., Fulford-Jones, T. R. F., Nawoj, A., Clavel, A., Shnayder, V., Mainland, G., Welsh, M., and Moulton, S. (2004). Sensor Networks for Emergency Response: Challenges and Opportunities. *IEEE Pervasive Computing*, 3(4):16–23.

[Loubser, 2006] Loubser, M. (2006). Delay Tolerant Networking for Sensor Networks. Technical Report Technical Report T2006-01, SICS - Swedish Institute of Computer Science.

[Lu and Lau, 1996] Lu, C. and Lau, S.-M. (1996). An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple Task Classes. In *Proceedings of the* 16^{th} *annual IEEE International Conference on Distributed Computing Systems (ICDCS 1996)*, pages 629–636.

[Madden et al., 2002] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. (2002). TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the* 5^{th} *annual ACM Symposium on Operating Systems Design and Implementation (OSDI 2002)*, pages 131–146.

[Madden et al., 2005] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. (2005). TinyDB: an Acquisitional Query Processing System for Sensor Networks. *Transactions on Database Systems, ACM Press*, 30(1):122–173.

[Mainwaring et al., 2002] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. (2002). Wireless Sensor Networks For Habitat Monitoring. In *Proceedings of the* 1^{st} *annual ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, pages 88–97.

[Manjhi et al., 2005] Manjhi, A., Nath, S., and Gibbons, P. B. (2005). Tributaries and Deltas: Efficient and Robust Aggregation in Sensor Network Stream. In *Proceedings of the annual ACM Special Interest Group on Management of Data (SIGMOD 2005)*.

[McCartney and Sridhar, 2006] McCartney, W. P. and Sridhar, N. (2006). Abstractions for safe concurrent programming in networked embedded systems. In *Proceedings of the* 4^{th} *annual ACM International Conference on Embedded Networked Sensor Systems (SenSys 2006)*, pages 167–180.

[Melodia et al., 2006] Melodia, T., Pompili, D., and Akyldiz, I. (2006). A Communication Architecture for Mobile Wireless Sensor and Actor Networks. In *Proceedings of the* 3^{rd} *annual IEEE International Conference on Sensor, Mesh and Ad hoc Communications and Networks (SECON 2006)*.

[Melodia et al., 2007] Melodia, T., Pompili, D., and Akyldiz, I. (2007). Communication and Coordination in Wireless Sensor and Actor Networks. *IEEE Transactions on Mobile Computing*, 6(10):1116–1129.

[MESA, 2010] MESA (2010). `http://www.projectmesa.org/`.

[MICAz, 2010] MICAz (2010). `http://www.xbow.com/Products/productdetails.aspx?sid=164`.

[Moh et al., 1998] Moh, W., Dongming, Y., and Makki, K. (1998). Wireless LAN: study of hidden-terminal effect and multimedia support. In *Proceedings of the* 7^{th} *annual International Conference on Computer Communications and Networks (ICCCN 1998)*, pages 422–431.

[Moteiv, 2006] Moteiv (2006). TMote Sky - Ultra low power IEEE 802.15.4 compliant wireless sensor module. Technical Report Technical Report - Moteiv.

[Müller, 2008] Müller, K. G. (2008). SimPy, a discrete event simulation package in Python. In *Proceedings of the* 1^{st} *Annual European Conference for Scientists using Python (EuroScipy 2008)*.

[Müller, 2010] Müller, K. G. (2010). SimPy. `http://simpy.sourceforge.net/`.

[Murty et al., 2008] Murty, R. N., Gosain, A., Tierney, M., Brody, A., Fahad, A., Bers, J., and Welsh, M. (2008). CitySense: An Urban-Scale Wireless Sensor Network and Testbed. In *Proceedings of the* 2^{nd} *annual IEEE Conference on Technologies for Homeland Security*, pages 583–588.

[Naik and Sivalingam, 2004] Naik, P. and Sivalingam, K. (2004). *A Survey of MAC Protocols for Sensor Networks*. Wireless Sensor Networks.

[Net, 2010] Net, D. (2010). `http://www.discovery-on-the.net/`.

[Ngai et al., 2006] Ngai, E. C. H., Lyu, M. R., and Liu, J. (2006). Real-Time Communication Framework for Wireless Sensor Actuator Networks. In *Proceedings of the IEEE annual Aerospace Conference*.

[Nitta et al., 2006] Nitta, C., Pandey, R., , and Ramin, Y. (2006). Y-threads: Supporting concurrency in wireless sensor networks. In *Proceedings of the* 2^{nd} *annual IEEE/ACM International Conference on Distributed Computing on Sensor Systems (DCOSS 2006)*, pages 169–184.

[Oh and Lee, 2008] Oh, S. J. and Lee, C. W. (2008). u-Healthcare SensorGrid Gateway for connecting wireless sensor network and grid network. In *Proceedings of the* 10^{th} *annual IEEE International Conference on Advanced Communication Technology (ICACT 2008)*, pages 827–831.

[Österlind, 2006] Österlind, F. (2006). A Sensor Network Simulator for the Contiki OS. Technical Report Technical Report T2006-05, SICS - Swedish Institute of Computer Science.

[Owen et al., 1998] Owen, M., Galea, E., Lawrence, P., and Filippidis, L. (1998). AASK - Aircraft Accidents Statistics and Knowledge: a Database of Human Experience in Evacuation, Derived from Aviation Accident Reports. In *Proceedings of the* 1^{st} *International Symposium on Human Behaviour in Fire*, pages 509–518.

[Park and Srivastava, 2003] Park, H. and Srivastava, M. B. (2003). Energy-Efficient Task Assignment Framework for Wireless Sensor Networks.

[Park et al., 2003] Park, J., Park, S., Kim, D., Cho, P., and Cho, K. (2003). Experiments on radio interference between wireless LAN and other radio devices on a 2.4 GHz ISM band. In *Proceedings of the* 57^{th} *IEEE Semiannual Vehicular Technology Conference (VTC 2003)*, pages 1798–1801.

[Pasztor et al., 2010] Pasztor, B., Mottola, L., Mascolo, C., Picco, G. P., Ellwood, S., and Macdonald, D. (2010). Selective Reprogramming of Mobile Sensor Networks through Social Community Detection. In *Proceedings of the* 7^{th} *annual European Conference on Wireless Sensor Networks (EWSN 2010)*, pages 178–193.

[Petriu et al., 2000] Petriu, E. M., Georganas, N. D., Petriu, D. C., Makrakis, D., and Groza, V. Z. (2000). Sensor-Based Information Appliances. *IEEE Instrumentation and Measurement Magazine*, 3(4):31– 35.

[Polastre et al., 2004] Polastre, J., Hill, J., and Culler, D. (2004). Versatile Low Power Media Access for Wireless Sensor Networks. In *Proceedings of the* 2^{nd} *annual ACM International Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pages 95–107.

[Pontelli et al., 2010] Pontelli, E., Le, H. V., and Son, T. C. (2010). An investigation in parallel execution of answer set programs on distributed memory platforms: Task sharing and dynamic scheduling. *Computer Languages, Systems and Structures*, 36(2):158–202.

[Proakis, 2000] Proakis, J. (2000). *Digital Communications*. McGraw-Hill Science, Engineering, Math.

[Qi et al., 2003] Qi, H., Xu, Y., and Wang, X. (2003). Mobile-Agent-Based Collaborative Signal and Information Processing in Sensor Networks. In *Proceedings of the IEEE Special Issue on Sensor Networks and Applications*, volume 91, pages 1172–1183.

[Qin et al., 2009] Qin, X., Jiang, H., Manzanares, A., Ruan, X., and Yin, S. (2009). Dynamic Load Balancing for I/O-Intensive Applications on Clusters. *ACM Transactions on Storage (TOS)*, 5(3):1–38.

[RadioCooja, 2010] RadioCooja (2010). A Ray-Tracing Based Radio Medium in Cooja. `http://www.sics.se/contiki/news/a-ray-tracing-based-radio-medium-in-cooja.html`.

[Raimondi et al., 2008] Raimondi, F., Skene, J., and Emmerich, W. (2008). Efficient Online Monitoring of Web-Service SLAs. In *Proceedings of the* 16^{th} *ACM International Symposium on the Foundations of Software Engineering (SIGSOFT/FSE 2008)*, pages 170–180.

[Rajendran et al., 2003] Rajendran, V., Obraczka, K., and Garcia-Luna-Aceves, J. J. (2003). Energy-effcient collision-free medium access control for wireless sensor networks. In *Proceedings of the* 1^{st} *annual ACM International Conference on Networked Sensor Systems (SenSys 2003)*, pages 181–192.

[Rajendran et al., 2006] Rajendran, V., Obraczka, K., and Garcia-Luna-Aceves, J. J. (2006). Energy-effcient, collision-free medium access control for wireless sensor networks. *Wireless Networks*, 12(1):63–78.

[Reddy et al., 2009] Reddy, A. M. V., Kumar, A. P., Janakiram, D., and Kumar, G. A. (2009). Wireless sensor network operating systems: a survey. *International Journal of Sensor Networks*, 5(4):236–255.

[Rondini and Hailes, 2007a] Rondini, E. and Hailes, S. (2007a). A Contiki-based Prototype for Creating Wireless Ad Hoc Grids. In *Proceedings of the* 1^{st} *Contiki Hands-On Workshop*.

[Rondini and Hailes, 2007b] Rondini, E. and Hailes, S. (2007b). Distributed Computation in Wireless Ad Hoc Grids with Bandwidth Control. In *Proceedings of the* 5^{th} *annual ACM Conference on Embedded Networked Sensor Systems (SenSys 2007)*, pages 437–438.

[Rondini and Hailes, 2010] Rondini, E. and Hailes, S. (2010). Empirical Study of Bandwidth-Aware Distributed Wireless Ad-hoc Grids. *ACM Transactions on Computer Systems (Under submission)*.

[Rondini et al., 2008a] Rondini, E., Hailes, S., and Li, L. (2008a). Distributed Wireless Ad hoc Grids With Bandwidth Control For Collaborative Node Localisation. In *Proceedings of the* 27^{th} *annual IEEE/Boeing Military Communications Conference (MILCOM 2008)*, pages 1–8.

[Rondini et al., 2008b] Rondini, E., Hailes, S., and Li, L. (2008b). Load Sharing and Bandwidth Control in Mobile P2P Wireless Sensor Networks. In *Proceedings of the* 5^{th} *annual IEEE International Workshop on Mobile Peer-to-Peer Computing (MP2P 2008) in conjunction with the* 6^{th} *IEEE International Conference on Pervasive Computing and Communication (PerCom 2008)*, pages 468–473.

[Ruiz-Ibarra and Villasenor-Gonzalez, 2008] Ruiz-Ibarra, E. and Villasenor-Gonzalez (2008). Cooperation Mechanism Taxonomy for Wireless Sensor and Actor Networks. *Ad Hoc and Sensor Wireless Networks*, 7:91–113.

[SAFECOM, 2010] SAFECOM (2010). `http://www.safecomprogram.gov/SAFECOM/`.

[Sensoy et al., 2009] Sensoy, M., Le, T., Vasconcelosi, W. W., Norman, T. J., and Preece, A. D. (2009). Resource Determination and Allocation in Sensor Networks: A Hybrid Approach. *The Computer Journal*.

[Sentilla, 2010] Sentilla (2010). `http://www.sentilla.com/`.

[SETI@Home, 2010] SETI@Home (2010). http://setiathome.berkeley.edu/.

[Shah et al., 2006] Shah, G. A., Bozyigit, M., Akan, O. B., and Baykal, B. (2006). Real Time Coordination and Routing in Wireless Sensor and Actuator Networks. In *Proceedings of the* 6^{th} *annual Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN 2006)*.

[Shah et al., 2009] Shah, G. A., Bozyigit, M., and Hussain, F. B. (2009). Cluster-based coordination and routing framework for wireless sensor and actor networks. *Wireless Communications and Mobile Computing*.

[Shang and Ruml, 2004] Shang, Y. and Ruml, W. (2004). Improved MDS Based Localization. In *Proceedings of the* 23^{d} *annual IEEE Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*.

[Shang et al., 2003] Shang, Y., Ruml, W., Zhang, Y., and Fromherz, M. (2003). Localization from Mere Connectivity. In *Proceedings of the* 4^{th} *annual ACM International Symposium on Mobile ad Hoc Networking and Computing (MobiHoc 2003)*.

[Shang et al., 2004] Shang, Y., Ruml, W., Zhang, Y., and Fromherz, M. (2004). Localization from Connectivity in Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(11):961–974.

[Shi and Kencl, 2006] Shi, W. and Kencl, L. (2006). Sequence-preserving adaptive load balancers. In *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 143–152.

[Shina et al., 2007] Shina, S. Y., Parkb, H. S., and Kwona, W. H. (2007). Mutual interference analysis of IEEE 802.15.4 and IEEE 802.11b. *Computer Networks*, 51(12):3338–3353.

[Simulator, 2010] Simulator, N. (2010). http://www.isi.edu/nsnam/.

[Singh and Prasanna, 2003] Singh, M. and Prasanna, V. K. (2003). A Hierarchical Model for Distributed Collaborative Computation in Wireless Sensor Networks. In *Proceedings of the* 17^{th} *annual IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2003)*, pages 166–176.

[SMARTFIRE, 2010] SMARTFIRE (2010). http://fseg.gre.ac.uk/smartfire/index.html.

[Stojmenovic, 2005] Stojmenovic, I. (2005). *Handbook of Sensor Networks: Algorithms and Architectures*. Wiley Interscience on Parallel and Distributed Computing.

[SUAAVE, 2010] SUAAVE (2010). http://www.suaave.org/.

[SunSPOT, 2010] SunSPOT (2010). http://sunspotworld.com/.

[System, 2010a] System, C. O. (2010a). http://www.sics.se/~adam/contiki/.

[System, 2010b] System, T. O. (2010b). `http://www.tinyos.net/`.

[Taylor et al., 1996] Taylor, S., Galea, E. R., Patel, M. K., Petridis, M., Knight, B., and Ewer, J. (1996). SMARTFIRE: An Intelligent Fire Field Model. In *Proceedings of the* 7^{th} *International Fire Science and Engineering Conference (Interflam 1996)*, pages 671–680.

[TELOSB, 2010] TELOSB (2010). `http://www.xbow.com/Products/productdetails.aspx?sid=252`.

[Tham and Buyya, 2005] Tham, C.-K. and Buyya, R. (2005). SensorGrid: Integrating Sensor Networks and Grid Computing. *CSI Communications, Computer Society of India*, 29(1):24–29.

[Tipsuwan et al., 2009] Tipsuwan, Y., Kamonsantiroj, S., Srisabye, J., and Chongstitvattana, P. (2009). An auction-based dynamic bandwidth allocation with sensitivity in a wireless networked control system. *Computers and Industrial Engineering*, 57(1):114–124.

[Tuy and Muyal, 2009] Tuy, B. and Muyal, S. (2009). Report on IPv6 Network Services and Applications deployed within GEANT and ERNET communities. Technical Report ICT-2007-223804-WP3-D3.1-v2.

[U-2010, 2010] U-2010 (2010). `http://u2010.eu/index.php`.

[UNICORE, 2010] UNICORE (2010). `http://www.unicore.eu/`.

[van Dam and Langendoen, 2003] van Dam, T. and Langendoen, K. (2003). An Adaptive energy-effcient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the* 1^{st} *annual ACM International Conference on Networked Sensor Systems (SenSys 2003)*, pages 171–180.

[van Willigen et al., 2009] van Willigen, W. H., Neef, R. M., van Lieburg, A., and Schut, M. C. (2009). WILLEM: A Wireless InteLLigent Evacuation Method. In *Proceedings of the* 3^{rd} *annual IARIA International Conference on Sensor Technologies and Applications (SENSORCOMM 2009)*, pages 382–387.

[Varakliotis et al., 2009] Varakliotis, S., Stephan, N., and Kirstein, P. T. (2009). Open Multi-Purpose Gateway for Emergency Services Internetworking. In *Proceedings of the 2009 annual IEEE International Conference on Communications (ICC Workshops 2009)*, pages 1–6.

[Vinyals et al., 2008] Vinyals, M., Rodrguez-Aguilar, J. A., and Cerquides, J. (2008). A Survey on Sensor Networks from a Multi-Agent Perspective. In *Proceedings of the* 2^{nd} *annual International Workshop on Agent Technology for Sensor Networks (ATSN 2008)*.

[Welsh and Mainland, 2004] Welsh, M. and Mainland, G. (2004). Programming sensor networks using abstract regions. In *Proceedings of the* 1^{st} *annual Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pages 3–3.

[Xiong et al., 2010] Xiong, N., He, J., Yang, Y., He, Y., hoon Kim, T., and Lin, C. (2010). A Survey on Decentralized Flocking Schemes for a Set of Autonomous Mobile Robots. *Journal of Communications*, 5(1):31–38.

[Xu and Qi, 2004] Xu, Y. and Qi, H. (2004). Distributed Computing Paradigms for Collaborative Signal and Information Processing in Sensor Networks. *Journal of Parallel and Distributed Computing, ACM Press*, 64(8):945–959.

[Yao and Gehrke, 2002] Yao, Y. and Gehrke, J. (2002). The Cougar Approach to In-Network Query Processing in Sensor Networks. In *Proceedings of the* 21^{th} *annual ACM SIGMOD International Conference on Management of Data*, pages 9–18.

[Ye and Heidemann, 2003] Ye, W. and Heidemann, J. (2003). Medium Access Control in Wireless Sensor Networks. Technical Report Technical Report ISI-TR-580, USC/Information Sciences Institute.

[Ye et al., 2002] Ye, W., Heidemann, J., and Estrin, D. (2002). An energy-effcient MAC protocol for Wireless Sensor Networks. In *Proceedings of the* 21^{st} *annual IEEE International Conference on Computer Communications (Infocom 2002)*, pages 1567–1576.

[Ye et al., 2004] Ye, W., Heidemann, J., and Estrin, D. (2004). Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks. *Transactions on Networking, IEEE/ACM Press*, 12(3):493–506.

[Yicka et al., 2008] Yicka, J., Mukherjeea, B., and Ghosal, D. (2008). Wireless Sensor Network Survey. *Journal of Computer Networks*, 52(12):2292–2330.

[Yuan et al., 2006] Yuan, H., Huadong, M., and Hongyu, L. (2006). Coordination Mechanism in Wireless Sensor and Actuator Networks. In *Proceedings of the* 1^{st} *annual International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2006)*.

[Zeng et al., 1998] Zeng, X., Bagrodia, R., and Gerla, M. (1998). GloMoSim: a library for parallel simulation of large-scale wireless networks. *ACM SIGSIM Simulation Digest*, 28(1):154–161.

[Zhao and Govindan, 2003] Zhao, J. and Govindan, R. (2003). Understanding Packet Delivery Performance In Dense Wireless Sensor Networks. In *Proceedings of the* 1^{st} *annual ACM International Conference on Embedded Networked Sensor Systems (SenSys 2003)*, pages 1–13.