Distributed Computing in Space-Based
Wireless Sensor Networks

A.H.M. Jallad

Submitted for the Degree of
Doctor of Philosophy
from the
University of Surrey

UniS

Surrey Space Centre
Faculty of Engineering and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

October 2008

© A.H.M. Jallad 2008
Summary

This thesis investigates the application of distributed computing in general and wireless sensor networks in particular to space applications. Particularly, the thesis addresses issues related to the design of "space-based wireless sensor networks" that consist of ultra-small satellite nodes flying together in close formations. The design space of space-based wireless sensor networks is explored. Consequently, a methodology for designing space-based wireless sensor networks is proposed that is based on a modular architecture. The hardware modules take the form of 3-D Multi-Chip Modules (MCM). The design of hardware modules is demonstrated by designing a representative on-board computer module. The on-board computer module contains an FPGA which includes a system-on-chip architecture that is based on soft components and provides a degree of flexibility at the later stages of the design of the mission.

The range of devices involved in space-based wireless sensor network environments inevitably leads to significant complexity in appropriately configuring, deploying, and dynamically reconfiguring the software. There is therefore a need for dedicated middleware platforms for space-based wireless sensor networks, with abstractions that can span the full range of heterogeneous systems, and which also offer consistent mechanisms with which to configure, deploy, and dynamically reconfigure both system and application level software.

The design of the middleware necessitates a comparison between two types of possible architecture: address-centric architecture and data-centric architecture. A comparison of both approaches is carried out in this thesis. A publish-subscribe mechanism is proposed for space-based wireless sensor networks that overcome the deficiencies of common wireless sensor networks publish-subscribe mechanisms such as Directed Diffusion in relation to space-based wireless sensor networks. The data-centric approach is then demonstrated using a middleware design that exhibits a component-based model in compliance with the general component-based space based wireless sensor networks philosophy.

Keywords: Wireless Sensor Networks, Distributed computing, Distributed Spacecraft Systems, formation-flying, data-centricity, FPGA

Email: a.jallad@surrey.ac.uk

WWW: http://www.ee.surrey.ac.uk/SSC
Acknowledgments

I firstly thank Almighty God for all his bounties and enabling me to continue this difficult journey towards receiving my PhD degree until the end. I would then like to thank my patient supervisor Dr. Tanya Vladimirova for her support during my time as her student. I want to thank her for providing me with not only technical support, but also more importantly the moral support that I needed during the difficult times of my studies.

I am also very thankful to the support I have got from my friends and brothers at Guildford. I would like in particular to mention the following brothers: Abdul-Aziz El-Gamouz and Salah Rashwan for the close brotherhood we had for the past few years; Abdussamad Ouwadaly and Djelloul Mahboob who I consider as my elder brothers; Hassan Mahmud, Azizun Azizal and Khaled Shawky for their valuable brotherhood. I want also to extend my thanks to my office mates for most of my time at the Space Centre, Juan Fernandez and Paul Blunt, for their good company.

Many thanks also go to my brothers Mahmoud and Sherif, and my aunt Hadeel for the love and affection they have provided me with, despite the geographical distances that have separated us during the past few years.

Last but not the least; I would like to express my deep thanks to my dearest parents. Their hard work and sacrifices have given me educational advantages that brought me to what I am in now. They had continued their faith in my capability during difficult times, while patiently waiting for my day to graduate. Without their inspiration and blessing, I would not have made it this far.
Contents

Summary ......................................................................................................................................... ii
Acknowledgments ........................................................................................................................... iii
Contents ........................................................................................................................................... iv
List of Figures .................................................................................................................................... viii
List of Tables ................................................................................................................................... x
List of Abbreviations ....................................................................................................................... xi

1 Introduction ................................................................................................................................... 1
  1.1 Motivation ........................................................................................................................... 2
  1.2 Scope and Objectives .......................................................................................................... 2
  1.3 Novelty and Contributions .............................................................................................. 3
  1.4 Publications ......................................................................................................................... 4
  1.5 Structure of the Thesis ......................................................................................................... 5

2 Literature Review .......................................................................................................................... 7
  2.1 Wireless Sensor Networks ................................................................................................... 7
    2.1.1 Applications of Wireless Sensor Networks ............................................................... 9
    2.1.2 Supporting Technologies ............................................................................................ 10
      2.1.2.1 Node Design ......................................................................................................... 11
      2.1.2.2 Operating System Design ..................................................................................... 12
      2.1.2.3 Middleware ........................................................................................................... 13
  2.2 Wireless Sensor Networks in Space .................................................................................. 13
  2.3 Distributed Spacecraft Missions ....................................................................................... 15
    2.3.1 Distributed Space Systems Terminology .................................................................... 16
    2.3.2 Review of Formation-Flying Missions ....................................................................... 17
    2.3.3 Satellite Miniaturization ............................................................................................ 19
  2.4 Approaches to Mission design ........................................................................................... 21
  2.5 Conclusions ....................................................................................................................... 22

3 Design Space Exploration of Space-Based Wireless Sensor Networks ...................................... 24
  3.1 Design Space of SB-WSN .................................................................................................. 24
    3.1.1 Deployment ................................................................................................................ 26
3.1.2 Mobility ...................................................................................................................... 26
3.1.3 Node Design Parameters ............................................................................................. 28
3.1.4 Heterogeneity .............................................................................................................. 31
3.1.5 Communication Modality ........................................................................................... 34
3.1.6 Infrastructure ............................................................................................................. 35
3.1.7 Network Topology ...................................................................................................... 35
3.1.8 Coverage ..................................................................................................................... 35
3.1.9 Connectivity ................................................................................................................ 36
3.1.10 Network Size ............................................................................................................. 36
3.1.11 Lifetime ....................................................................................................................... 36
3.1.12 Quality of Service requirements ................................................................................. 37
3.2 Module-Based Methodology to Designing SB-WSN ....................................................... 38
  3.2.1 MCM Design Flow ..................................................................................................... 40
  3.2.2 System-On-Chip Design Flow .................................................................................... 41
  3.2.3 Software Design Flow ................................................................................................. 43
    3.2.3.1 Operating System Analysis .................................................................................. 43
    3.2.3.2 Software Design Flow .......................................................................................... 47
3.3 Conclusions ....................................................................................................................... 48
4 Case-Study: Design of an On-Board Computer Module ..................................................... 50
  4.1 Review of On-Board Computer Miniaturization ............................................................... 50
  4.2 Module Design ................................................................................................................ 53
    4.2.1 Multi-Chip Module design ........................................................................................ 53
    4.2.2 System-On-Chip Design for the OBC Module ........................................................... 58
      4.2.2.1 Adding cores to the SoC ....................................................................................... 60
      4.2.2.2 SpaceWire for the SoC Design ............................................................................. 61
  4.3 Implementation Results .................................................................................................... 62
    4.3.1 SoC Resource Utilization ............................................................................................ 63
    4.3.2 OBC Performance ..................................................................................................... 65
  4.4 Conclusions ....................................................................................................................... 66
5 Data-Centricity for Space-Based Wireless Sensor Networks .................................................. 67
  5.1 Introduction to Data-Centric Networks .......................................................................... 67
    5.1.1 Approaches to Satellite Networking ......................................................................... 67
    5.1.2 Data-Centricity in Wireless Sensor Networks ............................................................ 68
    5.1.3 Data-Centricity for Wireless Sensor Networks ........................................................... 69
      5.1.3.1 Implementations of Data-Centric Networks ......................................................... 69
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.3.2</td>
<td>Data-Centric Routing</td>
<td>72</td>
</tr>
<tr>
<td>5.1.3.3</td>
<td>Data Aggregation</td>
<td>72</td>
</tr>
<tr>
<td>5.2</td>
<td>Experimental Evaluation</td>
<td>73</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Simulator Design</td>
<td>73</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Experimental Setup</td>
<td>76</td>
</tr>
<tr>
<td>5.2.2.1</td>
<td>The Publish/Subscribe Implementation: Directed Diffusion</td>
<td>76</td>
</tr>
<tr>
<td>5.2.2.2</td>
<td>The Client-server Implementation</td>
<td>78</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Impact of Mobility</td>
<td>79</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Scalability</td>
<td>82</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Discussion</td>
<td>82</td>
</tr>
<tr>
<td>5.3</td>
<td>Data-Centric Protocol Design</td>
<td>83</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Assumptions</td>
<td>83</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Protocol Design</td>
<td>84</td>
</tr>
<tr>
<td>5.3.2.1</td>
<td>Phase I: Initialization</td>
<td>85</td>
</tr>
<tr>
<td>5.3.2.2</td>
<td>Phase II: Interest Announcement and Gradient Setup</td>
<td>85</td>
</tr>
<tr>
<td>5.3.2.3</td>
<td>Phase III: Data Gathering</td>
<td>87</td>
</tr>
<tr>
<td>5.4</td>
<td>Data-Centric Protocol Experimentation</td>
<td>88</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Functionality Verification</td>
<td>88</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Performance Measurements</td>
<td>89</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Effect of Distances between the Nodes</td>
<td>90</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusions</td>
<td>91</td>
</tr>
<tr>
<td>6</td>
<td>Design of Middleware for Inter-Spacecraft Networking</td>
<td>93</td>
</tr>
<tr>
<td>6.1</td>
<td>Review on Middleware</td>
<td>93</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Introduction</td>
<td>93</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Middleware Classification</td>
<td>94</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Middleware for Wireless Sensor Networks</td>
<td>97</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Middleware for Space Applications</td>
<td>99</td>
</tr>
<tr>
<td>6.2</td>
<td>Middleware Design</td>
<td>100</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Design Requirements</td>
<td>100</td>
</tr>
<tr>
<td>6.2.2</td>
<td>The Architectural Design</td>
<td>102</td>
</tr>
<tr>
<td>6.3</td>
<td>Middleware Implementation</td>
<td>102</td>
</tr>
<tr>
<td>6.3.1</td>
<td>The Execution Environment</td>
<td>102</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Implementation Architecture</td>
<td>104</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Middleware Operation</td>
<td>107</td>
</tr>
<tr>
<td>6.4</td>
<td>Demonstration of Middleware Operation</td>
<td>110</td>
</tr>
<tr>
<td>6.5</td>
<td>Conclusions</td>
<td>113</td>
</tr>
</tbody>
</table>
7 Conclusions and Future Work ................................................................................................... 115
  7.1 Research Summary .......................................................................................................... 115
  7.2 Research Contributions ................................................................................................... 118
  7.3 Future Work .................................................................................................................... 119
    7.3.1 SB-WSN ................................................................................................................... 119
    7.3.2 Hardware Module Design ......................................................................................... 119
    7.3.3 Data-Centric Protocol for SB-WSN .......................................................................... 120
    7.3.4 MISA Middleware .................................................................................................... 121

Appendix – A: Miniaturization of Satellite Components and Systems ........................................ 123
  A.1 Enabling Technologies ...................................................................................................... 123
    A.1.1 MicroElectroMechanical Systems (MEMS) ............................................................... 123
    A.1.2 MultiChip Modules (MCM) ....................................................................................... 123
    A.1.3 Very Large Scale Integration (VLSI): ........................................................................ 124
  A.2 Miniaturised subsystems .................................................................................................... 124
    A.2.1 Propulsion ................................................................................................................... 124
    A.2.2 Attitude Determination and control (ADCS) .............................................................. 126
    A.2.3 Telemetry and Telecommand ..................................................................................... 127
    A.2.4 Thermal ....................................................................................................................... 129
    A.2.5 Power .......................................................................................................................... 130
    A.2.6 Memory ...................................................................................................................... 131

Appendix-B: Network Simulation Tools ...................................................................................... 132

Appendix – C: Derivation of Data-Centricity Equation ............................................................... 135

Bibliography ................................................................................................................................. 138
# List of Figures

Figure 2.1: Wireless sensor networks Venn diagram ............................................................. 8
Figure 2.2: The general architecture of a wireless sensor network ........................................ 8
Figure 2.3: Functional architecture of wireless sensor networks .......................................... 11
Figure 2-4: Structure of a typical mote ................................................................................ 12
Figure 2.5: Distributed Space System Venn Diagram .......................................................... 17
Figure 2.6: The trend in the number of spacecraft in formation-flying missions ................. 19
Figure 2.7: Co-orbiting satellite Assistant and the wafers that make its structure .............. 20
Figure 2.8: SSTL microsatellite modular structure ............................................................. 21
Figure 3.1: STK view of the formation flying configurations: leader-follower and same ground track ................................................................................................................. 27
Figure 3.2: STK simulation of the ranges between the satellites in a formation – leader-follower configuration ................................................................................................................. 28
Figure 3.3: STK simulation of the ranges between the satellites in a formation – same ground track configuration ................................................................................................................. 28
Figure 3.4: Node architecture of SB-WSN ........................................................................... 30
Figure 3.5: Drift between satellites due to external disturbances that affect the lifetime of the mission .......................................................................................................................... 33
Figure 3.6: SB-WSN general design flow ............................................................................. 39
Figure 3.7: 3D-Plus MCM Design flow ............................................................................. 42
Figure 3.8: System-on-Chip Hardware design flow ............................................................. 43
Figure 3.9: Software design flow for the nodes ................................................................. 47
Figure 3.10: Compilation of the application programs for the node software ................. 48
Figure 4.1: A picture of an MCM Module from 3D-Plus .................................................... 56
Figure 4.2: Mechanical design of the OBC MCM module ................................................. 57
Figure 4.3: Block diagram of the GR-PCI-XC2V3000 ....................................................... 58
Figure 4.4: Block Diagram of the System-On-Chip of the OBC Module ......................... 60
Figure 4.5: Simulation output of the writing and reading from register file added to the SoC ................................................................................................................................. 61
Figure 4.6: Block diagram of the SpaceWire CODEC ....................................................... 62
Figure 5.1: Data-Centric and Address-Centric approaches to distributed system design ... 68
Figure 5.2: Publish-Subscribe system ................................................................................ 70
Figure 5.3: Block Diagram of the SB-WSN Simulator ....................................................... 76
Figure 5.4: Three steps in directed diffusion (a) Interest propagation; (b) Gradient setup; (c) Data delivery along reinforced path ................................................................. 78
Figure 5.5: The basic client server interaction model and its socket-based derivative ...... 79
Figure 5.6: The success ratio of the distributed computing paradigms ............................. 81
Figure 5.7: The variation of the number of hops between the source and sink with the time of sending the subscription message ....................................................... 82
Figure 5.8: The scalability of the client-server and publish-subscribe systems .............. 83
Figure 5.9: Variation of the sampling time requirement with the accuracy of the GPS receiver and the distance between the nodes ......................................................... 84
Figure 5.10: The Interest Packet ..................................................................................... 86
Figure 5.11: Flowchart of the gradient set up process ..................................................... 87
Figure 5.12: The structure of the Data packet ................................................................. 88
Figure 5.13: Output of the simulator verifying the operation of the mechanism .............. 89
Figure 5.14: Effect of the distance between the nodes on the overhead percentage ......... 91
Figure 6.1: Middleware Layers ....................................................................................... 95
Figure 6.2: Classification of Middleware Software ......................................................... 96
Figure 6.3: The MISA middleware general architecture ... .......................... 103
Figure 6.4: Modules and interfaces of MISA ................................................................. 105
Figure 6.5: The state diagram of the middleware controller ........................................... 107
Figure 6.6: Interaction Diagram for sending a subscription (Interest) message .......... 108
Figure 6.7: Interaction Diagram for receiving a subscription (Interest) message by an intermediate node ..................................................................................... 109
Figure 6.8: Interaction Diagram for receiving a subscription (Interest) message by the source node ................................................................................................... 109
Figure 6.9: Interaction Diagram for receiving a data message by an intermediate node... 110
Figure 6.10: Interaction Diagram for receiving a data message by the sink node .......... 110
Figure 6.11: TOSSIM output for the MISA middleware operation ................................. 112
Figure 6.12: Snapshot of the GUI of the simulator during the simulations ..................... 113
Figure 7.1: Proposed extension to the SoC of the OBC module ..................................... 120
List of Tables

Table 1.1: Classification of satellite platforms based on mass ................................................................. 1
Table 2.1: Terrestrial applications of wireless sensor networks ................................................................. 10
Table 2.2: Sensor network platforms ........................................................................................................... 13
Table 2.3: Applications of wireless sensor networks in space ................................................................. 15
Table 3.1: Typical properties of the design space parameters of WSN ...................................................... 25
Table 3.2: Difference between Delta-V requirements for 400km Altitude .................................................... 34
Table 3.3: Comparison between the different space qualified interfaces .................................................... 40
Table 3.4: Comparison between the different types of operating systems ................................................ 45
Table 3.5: Comparison between RTEMS and TinyOS embedded operating systems ............................... 46
Table 4.1: State-Of-The-Art On-Board Computers .................................................................................. 51
Table 4.2: Comparison between Xilinx FPGAs ....................................................................................... 55
Table 4.3: Costs associated with MCM design ......................................................................................... 57
Table 4.4: Comparison between available soft core processors ............................................................... 59
Table 4.5: Implementation results ............................................................................................................ 63
Table 4.6: Size of Configuration files for the SoC Design ....................................................................... 64
Table 4.7: Memory Consumption of programs for the SoC Design ......................................................... 65
Table 4.8: Performance of the LEON processor ....................................................................................... 65
Table 5.1: Results from the functionality verification experiment ............................................................. 89
Table 5.2: Simulations settings for the evaluation experiments ................................................................. 90
Table 6.1: Middleware Classification ........................................................................................................ 98
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>Adaptive Communication Environment</td>
</tr>
<tr>
<td>ADCS</td>
<td>Attitude Determination and Control</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>CCSDS</td>
<td>Consultative Committee for Space Data Systems</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Requesting Broker Architecture</td>
</tr>
<tr>
<td>CORDIC</td>
<td>Co-Ordinate Rotation Digital Computer</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial-Off-The-Shelf</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>DSS</td>
<td>Distributed Space Systems</td>
</tr>
<tr>
<td>EDAC</td>
<td>Error Detection and Correction</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>FDIR</td>
<td>Fault Detection, Isolation and Recovery</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FPU</td>
<td>Floating Point Unit</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>HDLC</td>
<td>High-Level Data Link Control</td>
</tr>
<tr>
<td>IRIS</td>
<td>Intelligent Reconfigurable Integrated Satellite</td>
</tr>
<tr>
<td>JPL</td>
<td>Jet Propulsion Laboratory</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Access Network</td>
</tr>
<tr>
<td>MCM</td>
<td>Multi-Chip-Module</td>
</tr>
<tr>
<td>MOM</td>
<td>Message Oriented Middleware</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MMS</td>
<td>Magnetosphere Multi-Scale</td>
</tr>
<tr>
<td>NIT</td>
<td>Neighbour Information Table</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>OBC</td>
<td>On-Board Computer</td>
</tr>
<tr>
<td>OBDH</td>
<td>On-Board Data Handling</td>
</tr>
<tr>
<td>P-POD</td>
<td>Poly Picosatellite Orbital Deployer</td>
</tr>
<tr>
<td>PUS</td>
<td>Packet Utilization Standard</td>
</tr>
<tr>
<td>SB-WSN</td>
<td>Space-Based Wireless Sensor Network</td>
</tr>
<tr>
<td>SCOC</td>
<td>Spacecraft Controller On Chip</td>
</tr>
<tr>
<td>SDRAM</td>
<td>Synchronous Dynamic Random Access Memory</td>
</tr>
<tr>
<td>SEL</td>
<td>Single Event Latch-up</td>
</tr>
<tr>
<td>SoC</td>
<td>System-On-Chip</td>
</tr>
<tr>
<td>SPARC</td>
<td>Scalable Processor ARCHitecture</td>
</tr>
<tr>
<td>SSC</td>
<td>Surrey Space Centre</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
<tr>
<td>SSTL</td>
<td>Surrey Satellite Technology Limited</td>
</tr>
<tr>
<td>STK</td>
<td>Satellite Tool Kit</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>TID</td>
<td>Total Ionizing Dose</td>
</tr>
<tr>
<td>TTCP</td>
<td>Test Transport Control Protocol</td>
</tr>
<tr>
<td>TTL</td>
<td>Time To Live</td>
</tr>
<tr>
<td>TPF</td>
<td>Terrestrial Planet Finder</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
</tbody>
</table>
Chapter 1

1 Introduction

The continuous developments in the fields of VLSI and communications are fuelling factors for the increasing interest in the area of distributed computing. A distributed computing system is one that involves several computing entities connected with each other by some kind of network [1]. Distributed computing systems are more complex than systems that only involve single computing platforms. From here emerges the necessity of good distributed system design in order to manage the complexity of these systems [1]. While distributed computing is well established when applied to terrestrial applications, it remains to be under explored in space applications that include satellite platforms [2].

There are many ways to classify satellite platforms; one of which is to do it based on mass. A commonly used classification (that is adopted in this thesis) is taken from reference [3] and is shown in table 1.1. The term “Very-small satellites” is a term used to describe satellites that fall in the picosatellite and femtosatellite categories.

Table 1.1: Classification of satellite platforms based on mass

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Platform mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large Satellite</td>
<td>&gt;1000Kg</td>
</tr>
<tr>
<td>Medium sized satellite</td>
<td>&lt; 1000Kg</td>
</tr>
<tr>
<td>Mini-Satellite</td>
<td>&lt;500Kg</td>
</tr>
<tr>
<td>Micro-Satellite</td>
<td>&lt;100Kg</td>
</tr>
<tr>
<td>Nano-Satellite</td>
<td>&lt;10Kg</td>
</tr>
<tr>
<td>Pico-Satellite</td>
<td>&lt;1Kg</td>
</tr>
<tr>
<td>Femto-Satellite</td>
<td>&lt;0.1Kg</td>
</tr>
</tbody>
</table>
Another development within the space industry that is currently going side by side with miniaturization is formation-flying [4] [5]. Formation-flying was suggested in order to overcome the technological bottleneck that is experienced with miniaturization (as it depends on the advancements with related technologies such as VLSI and micromachining). Formation-Flying missions are missions that involve more than one satellite platform flying in close formations with one another, collaborating together in order to achieve common mission aims [6].

Wireless sensor networks (WSNs) have recently become an important technology within the area of distributed computing, finding way to several applications including habitat monitoring [7], intrusion detection [8] etc. The application of WSNs to space has been limited so far. This is considered normal due as applying advanced technology to space lags behind the state of the art (in many cases by more than a decade).

1.1 Motivation

The primary motivation behind this research work is the developments that are taking place in the space industry. These developments are two-fold: the miniaturization of satellites using advanced state-of-the-art technologies such as MicroElectroMechanical Systems (MEMS) [9] and spacecraft formation-flying gaining significant interest in the last decade with several formation flying missions being planned in the near future [4].

These two developments taking place in relation to space mission design have triggered the idea of having virtual satellite missions that consist of several (thousands) of tiny satellite nodes flying in close formations fulfilling a set of common mission aims. Several research projects have been initiated covering several aspects of such missions including formation flying control algorithms, inter-satellite links and micropropulsion [4]. However an area that has been under-looked so far is the role of data in the system and how it affects the design of such missions [2]. This leads us to the idea of looking at formation flying missions from a distributed computing point-of-view.

1.2 Scope and Objectives

The objective of the thesis is to investigate the possibility of deploying a distributed system for satellite missions that consist of ultra-small satellite nodes flying in close formations. In distributed system design, the nature of the application shapes the design of the distributed system. For satellite missions consisting of formation-flying ultra-small satellite node, the recently emerging field of wireless sensor networks (WSN) was found to
be the most suitable to apply to such missions; and hence those kind of missions have been renamed for the purpose of this thesis as Space-Based Wireless Sensor Networks (SB-WSN).

The approach to be used in exploring SB-WSNs is to identify the characteristic features by exploring the differences and similarities between SB-WSNs and other types of WSNs. The outcome of the investigation would be proposing a methodology to designing SB-WSN at the node-level and at the network-level. These methodologies need to be demonstrated practically by designing suitable hardware and software.

The objectives of this thesis are:

- To investigate the design of SB-WSN by identifying similarities and differences with terrestrial WSNs;
- To propose and verify a design methodology for node design of SB-WSNs;
- To investigate the network-level distributed computing component of SB-WSN.

1.3 Novelty and Contributions

Previous work reported in the literature mentioning SB-WSNs was found to be limited to three references: [10], [11] and [12]. In [10], Krishnamurthy et al. discusses SB-WSNs from a topology perspective. By topology they mean the communication pattern between the satellites in the formation. The emphasis of the work is on studying the influence of the topology over the stability of formations. The paper does not discuss the data handling part of SB-WSNs focusing on only one aspect of the design of SB-WSN namely, the topology.

In [11] and [12], Clare et al. cover another aspect of SB-WSNs. The focus of their work is on scheduling the communications resources to satisfy the sensor network traffic with minimum latency emphasizing layer 2 (link layer) issues. This is contrary to the work covered in this thesis which does not consider either the physical, link and MAC layers as this is the topic of several other researchers around the world dealing with inter-satellite communications protocols for satellite formations and constellations.

The work reported in this thesis is one of the first research works that have investigated the application of distributed computing to formation flying missions [2]. The initial ideas evolved to take the form of SB-WSN that consist of ultra-small satellite nodes flying in close formations. No previous work has been found in the literature that covers the design of SB-WSN. As this is a new area, there is a lot of scope to be discovered under this research title. We have therefore decided to focus our research on particular areas that we
feel are needed to form a foundation for future research to be based on. The specific novelty contributions of this research are as follows:

1. The architectural aspects of the design of a SB-WSNs are explored.
2. A Module-based design for nodes of SB-WSNs is proposed.
3. An evaluation of distributed computing architectures as applied to SB-WSNs is provided.
4. An on-board data handling architecture for the nodes of a SB-WSN is proposed.
5. A novel data-centric system based on the publish/subscribe mechanism to suitable for space environments, which takes into account the relative mobility of the satellite nodes is designed.
6. A new data-centric architecture specifically tailored to fit the needs of SB-WSNs is proposed, where the middleware design is based on the developed publish/subscribe mechanism.

1.4 Publications

The results of this research have been reported in the following publications:

Chapter 1: Introduction


1.5 Structure of the Thesis

This thesis is organised into 7 chapters.

Chapter 2 reviews the different components of the research that are involved in this thesis. These include satellite miniaturization and formation flying, distributed computing and middleware, and WSNs and their applications to space.

Chapter 3 investigates the design space of SB-WSN. The resemblance of WSNs to formation flying missions and how techniques used in the design of terrestrial WSNs apply to SB-WSN is highlighted in this chapter. The chapter also proposes a module-based approach to the design of the nodes for SB-WSNs. A methodology to designing SB-WSNs is detailed in the chapter.

Chapter 4 demonstrates the hardware part of the design methodology proposed in chapter 3 by designing an On-Board Computer module. It proves the possibility of designing single chip modules that may substitute for complete systems in larger spacecraft.

Chapter 5 provides an empirical evaluation of distributed computing approaches in relation to SB-WSNs. Based on the results of the evaluation, a data-centric publish/subscribe mechanism designed to overcome deficiencies in current state-of-the-art mechanisms is proposed in the same chapter. Experimental evaluation of this mechanism is given in the chapter.

Chapter 6 presents the design of a novel middleware software layer called MISA that is tailored for use on-board formation flying missions. The chapter serves two purposes: it transforms the data-centricity concept into an architectural design, and it demonstrates the design of software modules as part of the design methodology for SB-WSN that is proposed. The implementation of the middleware on a TinyOS platform is explained in the chapter.
Chapter 7. Summary and Conclusions, draws the various strands of the research effort together, concluding that a distributed system in space maybe constructed efficiently provided that appropriate design methodologies and protocols are adopted at both the node level and the network level. The same chapter also proposes ways in which this research work could be extended.
Chapter 2

2 Literature Review

This chapter provides a representative coverage of the related work from the literature. As the undertaken research is quite multidisciplinary, an abundance of material exists on related topics and to keep this chapter to a reasonable size, a representative overview of the various areas is presented. Further reviews specific to the different components of this research work are provided in the respective chapters. This literature review is divided into two sections: WSNs and related space-related aspects. The chapter is structured as follows. Section 2.1 reviews the area of WSNs. A survey on projects that involve the application of WSNs in space is given in section 2.2. Related work in the space domain, including formation-flying missions and satellite miniaturization are outlined in section 2.3. Section 2.4 reviews the current approaches to satellite mission design.

2.1 Wireless Sensor Networks

WSNs are a distinct type of distributed systems that consist of tiny nodes consisting of processing, sensing and communications, deployed on a large scale in order to achieve a common set of sensing aims [13] [14] [15]. The nodes are therefore,

1. Networked: The role of networking is to coordinate and perform higher-level tasks. This creates a basis for exploiting collaborative sensing and actuation

2. Embedded: Numerous embedded distributed devices enable monitoring and interaction with the physical world. The nodes are autonomous, small, and untethered systems.

3. Systems: The Sensing and actuation are tightly coupled to the physical world.

All of the above three characteristics (illustrated graphically in figure 2.1) also apply to space missions that consist of pico/femto-satellite networks flying in close formations. All the nodes are connected in a network via inter-satellite links on-board each node. A satellite is a complex embedded system that may be viewed as a multi-sensor, multi-actuator sensor node. This similarity between WSNs and formation flying satellite networks will be discussed further in chapter 3, section 3.1.
WSNs are a special type of ad hoc networks i.e. an infrastructure-less network [16]. By infrastructure-less we mean that the WSN has no fixed wired/wireless backbone [17]. There is no need centralised servers and no centralised administration. All of the nodes can act as routers. A global view of sensor networks is shown in figure 2.2. The sensor nodes are spread out in a sensor field [18]. Each of the sensor nodes in the network has the capability of collecting data and routing them back to the sink. The sink can be thought of as a root node in a tree structure that in some networks is designed to be a more capable node than the rest of the sensor nodes in the network. All the communications from the nodes in the network will be passed to the outside world through the sink node. Data are routed back to the sink via the wireless network multi-hop architecture and then through the sink. The sink may communicate with the operating node directly or remotely via an intermediate network (Internet or satellite) [18].
WSNs have triggered a large amount of research in the last decade in areas such as applications, node hardware design, operating systems architecture, communications algorithms and protocols [18]. In this section we present the current state-of-the-art on WSNs that is most relevant to this thesis.

2.1.1 Applications of Wireless Sensor Networks

One of the important factors that have stimulated research in WSNs is the large number of applications that they could be used for. A wide range of commercial and military applications of WSNs are identified. Hills [19] categorizes the applications of WSNs into three classes (with an additional category that is a hybrid of the three main ones) as follows:

- Environmental data collection
- Security Monitoring
- Node tracking
- Hybrid category

Applications of WSNs extend at different geographical scale levels. Health care applications, for example, are limited to the environmental premises that the patient would interact with. On the other side, environmental observation applications (pollution monitoring as an example) extends to include a wide geographical area. WSN applications also vary in the density of the network i.e. the number of nodes that are required for meeting the aims and goals of deploying the network.

Examples of terrestrial applications of WSNs are summarized in table 2.1. As can be seen from table 2.1, each network has a different set of aims and operates in a different type of environment (for example: ZebraNet in an animal habitat, Glacier in a sub-glacier environment, Ocean in oceans) which makes each network have a unique set of requirements and therefore a design that is specifically made to meet those set of requirements.

Following on from the conclusion that the design of WSNs are usually tailored to fit the application is will be used for, WSNs used for space require a specific of design requirements that are tailored to suite the space environment. This is in addition to the application-specific requirements that are entailed by that specific space application. These requirements will be studied in detail in section 3.1.
2.1.2 Supporting Technologies

The advances in WSNs depend very much on the developments that take place in the wide range of technologies that underpin their design, including hardware, system software and network communications. Figure 2.3 shows diagrammatically a functional view of WSNs. Figure 2.3 also illustrates the link between the different areas of research involved in the development of WSNs. In the figure the functions of the system is divided into several layers of abstraction with any other computer system. The functionality of the system extends across the different layers. Each of these layers is explained below with reference to WSNs.

Table 2.1: Terrestrial applications of wireless sensor networks

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Sensors</th>
<th>Size (number of nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Great Duck [20]</td>
<td>Observing the breeding behaviour of a small bird called Leach’s Storm Petrel.</td>
<td>Humidity, pressure, temperature, and ambient light level.</td>
<td>10s-100s</td>
</tr>
<tr>
<td>ZebraNet [21]</td>
<td>Observing the behaviour of wild animals within a spacious habitat</td>
<td>GPS receiver to obtain estimates of the position and speed of the animals</td>
<td>10s-100s</td>
</tr>
<tr>
<td>Glacier [22]</td>
<td>Monitoring of sub-glacier environments.</td>
<td>Pressure, temperature and tilt sensors</td>
<td>10s-100s</td>
</tr>
<tr>
<td>Ocean [23]</td>
<td>Obtaining a quantitative description of the state of the upper ocean and the patterns of ocean climate variability.</td>
<td>Temperature, salinity</td>
<td>1300</td>
</tr>
<tr>
<td>Vigilnet [24]</td>
<td>Ground surveillance. The general objective is to alert the military command and control unit in advance to the occurrence of events of interest in hostile regions.</td>
<td>Magnetometer</td>
<td>70</td>
</tr>
</tbody>
</table>
2.1.2.1 Node Design

WSNs research is usually centred on designing different system components to enable the system in general to operate for the longest possible period under extreme environments. This applies to different parts of the system including hardware, computer architecture, operating system, middleware and applications software. Constraints that are common in most WSNs are energy, computing power and memory. In addition, sensor networks should work unattended for long periods of time [25]. This implies that the nodes of the network must preserve their limited amount of energy for as long as possible. This makes power awareness the major concern for the hardware design of the individual nodes.

A common term that is now established within the WSNs research community is the term wireless “mote”, which is a short name for “remote node”. The goal of the hardware of a mote is to provide computation, communications and storage in a single miniature device. The general architecture of a typical mote is shown in figure 2.4. The main components of the mote are the processing system, the communication system, the sensing system, the memory and a source of power. Additional optional capabilities may be incorporated into the mote design such as location finding system and a mobilizer for actuation purposes.
The TinyOS group at Berkeley has developed a hardware platform for WSNs [26]. Each device has limited power, computation, and storage resources. The devices do not contain any sensing capabilities but have interfaces to connect to external sensor boards. Table 2.2 lists the specifications of some Berkeley motes (Mica2, Mica2dot and Telos) in addition to other commercially available motes in order to provide an idea on the nature of motes designed for WSNs. From the table it can be observed that the majority of the surveyed motes use low-end processors such at Atmel ATMega 128L (8-bit) and TI MSP 430 (16-bit) in addition to small amounts of memory which are in the range of kilobytes. An exception to this is the Sensoria WINS mote which is designed for sensor networks that require motes with relatively high specification such as imaging.

### 2.1.2.2 Operating System Design

WSNs motes described above are obviously limited in their processing power, energy and memory. Currently available Commercial Off-The-Shelf (COTS) embedded operating systems are not suitable for use in WSNs [32] [33]. This is primarily because COTS embedded operating systems do not consider the requirements of WSN operating system design derived from the high constraints imposed by the limitation in resources. This has lead to operating system design for WSNs becoming one of the active research topics within the WSN research community.

One of the leading projects in the field of WSNs operating system design is the TinyOS project [33]. The TinyOS component-based and event-driven execution model enables fine-
grained power management and yet allows some scheduling flexibility that is necessary due to the unpredictable nature of wireless communication and the physical world interfaces. However, TinyOS is unable to support multimodal tasking well, and moreover, it does not provide real-time scheduling and thus it is not suitable for real-time sensor network systems.

Table 2.2: Sensor Network Platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Processor</th>
<th>RAM</th>
<th>ROM</th>
<th>Radio</th>
<th>Actuators</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mica2 [26]</td>
<td>Atmel ATMega 128L</td>
<td>4kB</td>
<td>128kB</td>
<td>CC1000</td>
<td>extensible</td>
<td>Extensible</td>
</tr>
<tr>
<td>MicaZ [27]</td>
<td>Atmel ATMega 128L</td>
<td>4kB</td>
<td>128kB</td>
<td>CC1000</td>
<td>3 LEDs, speaker</td>
<td>Extensible</td>
</tr>
<tr>
<td>BT-Nodes [28]</td>
<td>Atmel ATMega 128L</td>
<td>4kB</td>
<td>128kB</td>
<td>ZV4002</td>
<td>4 LEDs</td>
<td>Extensible</td>
</tr>
<tr>
<td>Telos [29]</td>
<td>TI MSP 430</td>
<td>2 KB</td>
<td>48 KB</td>
<td>CC2420</td>
<td>3 LEDs</td>
<td>Humidity,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>temperature,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>light, 2 buttons</td>
</tr>
<tr>
<td>Sensoria WINS</td>
<td>TMS 320V5502</td>
<td>64 MB</td>
<td>32 MB</td>
<td>802.11</td>
<td>—</td>
<td>GPS</td>
</tr>
<tr>
<td>Cricket [31]</td>
<td>Atmel ATMega 128L</td>
<td>4 KB</td>
<td>128 KB</td>
<td>CC1000</td>
<td>3 LEDs</td>
<td>Ultrasound</td>
</tr>
</tbody>
</table>

2.1.2.3 Middleware

Middleware is the software that resides between the operating system and the application [1]. It is used to provide functional components that are not typically provided by the operating system. Typical services offered by middleware software include time synchronization [33], group management [34], localization [35], and node discovery [36]. It also provides a higher level of abstraction for the application programmer simplifying software development. Middleware software is explained in detail in chapter 6 with a comprehensive survey provided in section 6.1.

2.2 Wireless Sensor Networks in Space

The application of WSNs in space can be classified in one of four different categories of possible applications as follows [37] [38]:
Chapter 2: Literature Review

- **Microsensor networks (Sensor webs).**

Microsensor networks are the closest to the commonly known terrestrial WSNs in terms of network application, network architecture, node architecture and node size. Micosensor networks may consist of hundreds or thousands of microsensor motes that could be randomly deployed on a planet (such as Mars) or the moon. The European Space Agency (ESA) and other researchers have called these networked, collaborative collections of microsensor nodes “sensor webs” [39, 40]. Satellites and telescopes remotely "measuring" planets across the vast reaches of space allow large areas to be monitored. Unlike remote operations, sensor webs are placed inside the environment, thus making them capable of on-site detection which is not possible from afar.

- **Intra-spacecraft networks.**

Intra-spacecraft networks are those that deployed within the boundaries defined by the structure of the spacecraft. The motivations behind such intra-spacecraft wireless networks are manifold. Firstly, the wireless sensor nodes could reduce the weight and space used for cables that are used for data transmission. The wireless nature of such networks also leads to the simplification of the installation and maintenance of the network, because the need for additional cabling is minimized or eliminated. An example application of such networks is the monitoring of the structural health of large spacecrafts like the International Space Station (ISS).

- **Inter-vehicular Networks**

Inter-vehicular networks include space missions that involve multiple nodes which are interconnected by a wireless network. Examples are formation flying satellite missions and multi-robot missions. The nodes in inter-vehicular networks are characterized as being mobile.

- **Extra-Vehicular Activities Network**

Extra-vehicular activity (EVA) proximity networks are networks that aim at supporting operations that need to take place outside of a spacecraft. These operations may involve humans, manned vehicles or robots [39]. The nodes in these networks are mobile and generally involve a variety of sensors in the same network such as sensors measuring fluid transfer, blood pressure etc.

Table 2.3 shows the possible applications of each of the above categories of networks in space along with the typical features of the network. The sensor network that is the focus of this thesis comes under the third category i.e. inter-vehicular networks or missions that
consist of ultra-small satellite nodes flying in close formations. The related work on intervehicular networks is surveyed in the next section.

Table 2.3: Applications of wireless sensor networks in space

<table>
<thead>
<tr>
<th>Category</th>
<th>Applications</th>
<th>Engineering objectives</th>
<th>Wireless technology</th>
<th>Typical processor power, memory</th>
<th>Data-rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsensor networks</td>
<td>Fixed planetary micro-sensor web</td>
<td>Maximize data transfer per battery life</td>
<td>Zigbee</td>
<td>8-bit processors, kilobytes of memory</td>
<td>low</td>
</tr>
<tr>
<td>Intra-spacecraft networks</td>
<td>Spacecraft health monitoring, Astronaut health monitoring, General spacecraft transducer network</td>
<td>Maximize data transfer per battery life</td>
<td>Zigbee</td>
<td>variable</td>
<td>low</td>
</tr>
<tr>
<td>Inter-vehicular networks</td>
<td>Formation-flying networked spacecraft, Networked planetary surface robots.</td>
<td>Reliable mobile communications</td>
<td>CCSDS, IEEE802.11</td>
<td>32-bit processors, Megabytes of memory</td>
<td>medium</td>
</tr>
<tr>
<td>Extra-Vehicular activities networks</td>
<td>Tiny satellites orbiting a mother ship for inspection purposes</td>
<td>Maximize data transfer during the battery lifetime and reliable mobile communications</td>
<td>Zigbee</td>
<td>16/32 bit processor, Megabytes of memory</td>
<td>Variable (low to medium)</td>
</tr>
</tbody>
</table>

2.3 Distributed Spacecraft Missions

In this section a review of space research that is relevant to the design of SB-WSN is presented. Two research topics that are currently distinct from each other are merged together in the SB-WSN concept. These topics are formation flying and satellite miniaturization, which are detailed below.

The direct application of this research work is aimed at the space industry. Few other engineering environments are as demanding as space systems. Once launched, a spacecraft is on its own for as long as decade or more. Except in very specialized circumstances, such
as the Hubble telescope, repairs are out of the question. Consequently, the development environment is very challenging. The benefits of any new technology must be carefully weighed against the potential for introducing failures.

The harsh radiation environment makes space an unfriendly place for modern electronics. Any critical devices onboard the spacecraft must either be protected via shielding, or subjected to extensive and costly ‘radiation hardening’ or have a flight heritage record. Consequently, the state-of-the-art spacecraft processors are often a decade or more behind their terrestrial counterparts [41]. The dangers of radiation are not limited to the direct degradation of components. High-energy particles are known to flip bits in memory or microprocessor registers. Some actions can be taken to detect and correct most of these errors, but undetected problems of this type can be very serious.

### 2.3.1 Distributed Space Systems Terminology

Distributed space systems (DSS) are currently recognized as an important research topic that is expected to shape the future of space missions [42]. DSS are missions that involve more than one platform for the accomplishment of mission aims. The space community have quickly realised the advantages offered by distributed space missions and as a result a lot of money is being spent in the development of related technologies. While only a handful of DSS are now actually in space a large number of distributed space missions are planned for launch in the next decade (for examples and related references see section 2.3.2).

DSS can take several forms [43]. This has led to various terminologies describing the different forms of distributed space systems. Fig 2.5 shows the relationship between the different terms describing distributed space systems. The terminology described in the figure is described below [43]:

- A *distributed space system* is a system that consists of two or more satellites that are distributed in space and form a cooperative infrastructure for science measurement data acquisition, processing analysis and distribution. DSS do not need to link directly to other companion satellites and can be free to make independent observations. The following

- A *Sensor web* is a system of intra-communicating spatially distributed sensor crafts that may be deployed to monitor environments. Sensor webs may involve many non-space elements and are therefore not completely covered by DSS.
A group of satellites that have coordinated coverage, operating together under shared control, synchronised so that they overlap well in coverage and reinforce rather than interfere with other satellites' coverage is known as a satellite constellation.

A Cluster is a functional grouping of spacecraft, formations, or virtual satellites.

A formation is a multiple spacecraft system with desired position and/or orientation relative to each other or to a common target.

Formation flying is the term used for the tracking and maintenance of a desired relative separation, orientation or position between or among spacecraft.

A Virtual satellite is a spatially distributed network of individual satellites collaborating as a single functional unit, and exhibiting a common system-wide capability to accomplish a shared objective.

2.3.2 Review of Formation-Flying Missions

This section provides a brief survey of related formation-flying missions. In this thesis we focus on precise formation flying which consist of several small spacecrafts flying at close distances to each other. Formation-flying missions aim at using several small satellites to accomplish the tasks of a single complex spacecraft. Each satellite within a formation contributes to the overall objective of the system. If one were to fail, the integrity of the system would not be totally lost, as the remaining satellites would still function to achieve the mission objectives.

Clements et al. [44] classified formation-flying missions into three distinct classes depending on the rationale behind the mission:
- **Signal Separation**: Spatially distributed sensors on-board of different nodes in the formation collect measurements from the same source.

- **Signal Combination**: Distinct sensors on separate nodes collect data from different sources and merge this data on-board of the formation to extract global information of a particular phenomenon.

- **Signal Coverage**: A Sensor web with identical sensors on the nodes with the purpose of covering wide areas of surface (e.g. multi-point sensing). SB-WSN fall under the third category i.e. signal coverage or multipoint sensing.

Figure 2.6 shows the summary of the outcome of the survey of planned formation-flying missions. The figure shows that the number of spacecraft per formation-flying mission is (almost exponentially) increasing with time. This is a primary motivation behind research into SB-WSN design. The Earth-Observation (EO) mission that has been launched in the year 2000 illustrates a simple form of formation flying. The EO-1 satellite will fly in the same ground track with Landsat-7 but several minutes behind it. The EO-1 satellite has three earth observing instruments that are compared on earth against the earth observing capability of LandSat-7 [45].

ST-5 is a technology demonstration mission that was successfully launched in 2006 [46]. The mission consists of three identical microsatellites each having a total mass of 25Kg. The main aim of the mission is to act as a demonstrative proof-of-concept mission for future missions consisting of a larger number of nodes. The Magnetosphere MultiScale (MMS) Mission aims at studying the earth's magnetosphere by deploying four identical spacecraft in a tetrahedral formation [47]. MMS is a typical example of a multipoint sensing SB-WSN. A similar mission that is also aimed at studying the Earth's magnetosphere with a formation of 4 satellites is the SWARM mission [48].

NASA proposed the Terrestrial Planet Finder (TPF) mission with the aim of detecting extra solar terrestrial planets [49]. TPF and other interferometry missions are examples of the signal separation class of formation-flying missions. Other examples of interferometry missions include MAXIM [50], Darwin [51], Stellar Imager [52] and Planet Imager [53].
2.3.3 Satellite Miniaturization

Miniaturization of satellites has recently emerged as an active area of research that includes two different domains: miniaturization at the spacecraft level and miniaturization at the subsystem and component level. At the spacecraft level, the greatest amount of research projects has come under the category of pico-satellites or CubeSats [54]. One of the most commonly used definitions of CubeSats is [55]: A cube sat is a satellite that has dimensions of 10cmX10cmX10cm and weighs less than 1kg.

Most of existing CubeSats have been developed by academic institutions for educational purposes such as DTUSAT [56], YAMSAT [57], and MEROPE [58]. Due to the academic nature of these projects, the relatively small budgets allocated to these projects and the tight mass and volume constraints they have, it can be said that these satellites trade performance for mass and volume. As a result the industrial application of these satellites has so far been very limited.

A different kind of project was carried out by the US Aerospace Corporation – the co-orbiting Satellite Assistant (COSA) [59]. COSA has dimensions 5cmX5cmX5cm and is intended to assist a larger mother ship, such as the Space Shuttle or International Space Station, by providing an external set of free-flying sensors. This amount of miniaturization is achieved through a pulsed UV laser volumetric direct-patterning technique to fabricate the structural members and key fluidic distribution systems of the satellite.
The vehicle is fashioned out of 7 laser patterned wafers, electronics boards and a battery. The electronics portion of the COSA vehicle includes a wireless communication system, 2 micro-controllers for system control and a MEMS gyro for relative attitude determination. The COSA vehicle is designed to be mass producible. COSA is shown in figure 2.7.

The other dimension in spacecraft miniaturization is the miniaturization of the spacecraft subsystems and components. The projects that deal with various spacecraft subsystems and components miniaturization are quite numerous and are therefore detailed in Appendix A. However, a factor that is common to all of these projects is the enabling technologies which they stand on. For example a technology that is relatively new is the Micro-Electro-Mechanical Systems (MEMS) technology [9]. Complete mechanical and electrical systems can be incorporated into a single silicon module. MEMS are micron-to-millimetre scale sensors and actuators that can be produced using modified semiconductor fabrication techniques which provide miniaturization, multiplicity and microelectronics. Microelectronics provides intelligence by merging sensors, actuators and logic together forming closed-loop feedback components and systems. Various spacecraft subsystems and components have been miniaturized using the MEMS technology including accelerometers [60], control moment gyroscopes [61], micropropulsion systems [62][63], magnetometers [64], sun sensors [56][65] and heat pipes [66].

Another enabling technology is system-on-chip (SoC). SoC design is the next step in the technology evolution that represents the combined tools and methodology to effectively utilize the continuously increasing area of VLSI chips, through the development of very large complex systems, on a single silicon substrate, in a very short design cycle. The integration side of SoC design starts with partitioning of the system around the primarily pre-existing, block-level functions and identifying the new or differentiating functions.
needed" [78]. System-On-a-Chip design is much more than high-level integration of IP cores such as microprocessors, memory and peripherals. It requires "system expertise" and "system know-how" in order to maximize the effect of translating system functionality to a single-chip implementation. SoC designs are typically either derivative designs with increased functionality, or convergence designs where previously separate functions are integrated [78]. The System-on-chip technology is used in the proposal for the on-board computer design of the data handling system of a node within the satellite network as shown in chapter 4, section 4.2.

2.4 Approaches to Mission design

In this section the approach that has been proposed to mission design that are different from the traditional methodology of space mission design, are reviewed. Three approaches have been identified: SSTL's modular Small satellite design, Cubesat design and ChipSat design.

Surrey Satellite Technology Limited (SSTL) has been able to enter strongly the microsatellite market. SSTL have been able to do so for several reasons, one of which is its design practices which reduces the costs of the design to a large extent [67]. The other reason is the platform design of the satellites. The SSTL modular microsatellite (shown in figure 2.8) has no skeleton but rather a series of identical outline machined module boxes, stacked one on top of the other, to form a body onto which solar panels and instruments maybe mounted. The modules are held together by tie-rods that pass through the whole stack and allows some dissipation of vibrational energy [67].

![Figure 2.8: SSTL microsatellite modular structure [65]](image-url)
The microsatellite approach that is adopted by SSTL does provide a large level of flexibility in the design of satellite platforms. On the other end, there is the Satellite-on-Chip approach that is currently still in its initial research phases [59] [68] [69]. This approach aims at integrating a complete satellite on a single integrated chip. The advantage with this approach is that it offers a high level of miniaturization. However, there is a key problem with this approach; the difficulty and the high cost associated with designing these satellite platforms, despite the low cost of duplicating the platform. This problem makes the approach lack the design flexibility that is actually required for space mission design. This is because each satellite mission does usually have a unique set of aims that it needs to accomplish.

The third approach, namely the cubesat design, cannot in fact be considered a design approach by itself that may be compared with the ones mentioned above. This is because there is no specific design methodology associated with the approach except for keeping to the external standards of the platform, e.g. 1Kg mass, 10cm3 dimensions etc.

In this thesis a module-based approach is proposed in the next chapter, which combines a middle point between the SSTL microsatellite approach and the satellite-on-chip approach, while being able to integrate swiftly with the cubesat approach.

2.5 Conclusions

This chapter provided a review of the literature on the main topics related to SB-WSN. Further reviews on the other topics explored in the thesis are covered in the respective chapters. SB-WSNs are basically formation-flying space missions that consist of ultra-small satellite nodes. There currently exist no SB-WSN in space and therefore the approach used in the review was to explore the area of WSNs in general and its current applications in the space domain extracting from those experiences ideas that could be used in our investigation.

The review showed that WSNs are deeply explored for terrestrial applications. However, the application of WSNs in space has not yet been fully exploited yet. The most important conclusion from this chapter is that there is a good extend of similarity between WSNs and SB-WSN. This similarity is based on the following facts: that SB-WSN are multipoint sensing networks, the number of nodes in SB-WSN are in a continuous increase and continuous reduction in the physical mass and size of the individual nodes within the network. This provides a motivation for investigating the similarities and differences between WSNs in general and SB-WSN in further detail in the next chapter.
Chapter 2: Literature Review

The first section of the chapter provides a short survey of WSNs in general including terrestrial applications of WSN and the technologies associated with the advancement of this research area. The conclusion from the section is that many of the enabling technologies that are used for the development of WSNs are either already being used for space applications or they may be transferred to the space domain. This includes hardware miniaturization and software design practices. This thesis would attempt exploiting some of these technologies for SB-WSN.

The second section of the chapter showed the significant number of planned formation flying missions within the space industry. This shows the importance that formation flying will play in the future of space exploration. The reasons behind the continuous interest in formation flying are manifold: cheaper in cost, higher in reliability and capable of performing unprecedented mission aims. The trend in formation flying missions is to increase the number of nodes involved and reduce their sizes, complexity, cost and mass. This conclusion is in favour of SB-WSN research, as this shows the convergence of space formation flying missions towards SB-WSN.

An important aspect of SB-WSN is the miniaturization of the satellite platforms. The approaches to the design of such platforms have been reviewed. The outcome of the review was that there are three possible approaches to small satellite platform design: SSTL modular microsatellite approach, Satellite-on-chip approach and cubesat approach. While the modular microsatsellite approach provides flexibility it lacks the miniaturization required for designing SB-WSN nodes. On the other hand, the satellite-on-chip approach provides a high-level of miniaturization but lacks design flexibility at mission design time. A module-based approach that could be integrated with cubesats to provide a middle point between the two extreme approaches is proposed in the next section.
Chapter 3

3 Design Space Exploration of Space-Based Wireless Sensor Networks

This chapter provides a detailed design space exploration study of formation-flying missions consisting of ultra-small satellite nodes. Such networks are treated as a unique type of WSNs and are referred to as Space-Based Wireless Sensor Networks. Due to the wide range of applications that the concept of WSNs have found its way into, with each application having a different set of characteristic features that are reflected in the design, an extensive design space with many dimensions are involved in the design.

The purpose of this chapter is to investigate the design space of SB-WSNs in order to be able to identify the differences between SB-WSNs and other types of WSNs. It provides a detailed study of the general features of SB-WSNs that is the starting point to making the design choices of the node-level design discussed in chapter 4 and the network-level design discussed in chapter 5. The chapter is divided into two sections. Section 3.1 explores the design space of SB-WSN. Section 3.2, on the other hand, proposes a novel methodology for the design of SB-WSN that is based on a modular approach.

3.1 Design Space of SB-WSN

The applications of WSNs form an extensive design space with many dimensions, such as the following [70].

- Deployment
- Mobility
- Cost, size, resources, and energy
- Heterogeneity
- Communication modality
- Infrastructure
- Network topology
Chapter 3: Design Space Exploration of Space-Based Wireless Sensor Networks

- Coverage
- Connectivity
- Network size
- Lifetime
- Other quality of service requirements

Table 3.1 presents typical properties of the WSN parameters listed above. In this section we will apply the above design factors to SB-WSNs. Such an approach helps in highlighting the differences between SB-WSNs and 'mainstream' WSNs and in understanding the unique features of SB-WSN. The design space of SB-WSN is discussed in the following subsections.

Table 3.1: Typical properties of the design space parameters of WSN

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment</td>
<td>Onetime, incremental or as random activity</td>
</tr>
<tr>
<td>Mobility</td>
<td>Occasional or continuous performed by either selected or all nodes.</td>
</tr>
<tr>
<td>Resources</td>
<td>Very resource limited to unlimited. Resources include cost, size, memory and energy</td>
</tr>
<tr>
<td>Heterogeneity</td>
<td>A single type of node or diverse sets of differing properties and hierarchies</td>
</tr>
<tr>
<td>Communication modality</td>
<td>Apart from radio frequency, optical, acoustic, inductive and capacitive coupled communication have been used</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Different applications exclude, permit or require the use of fixed infrastructure</td>
</tr>
<tr>
<td>Network topology</td>
<td>Single hop, star, multihop, mesh and/or multitier</td>
</tr>
<tr>
<td>Coverage</td>
<td>Sparse, dense or redundant</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Continuous, occasional or sporadic</td>
</tr>
<tr>
<td>Network size</td>
<td>Ranging from tens of nodes to thousands</td>
</tr>
<tr>
<td>Lifetime</td>
<td>Few hours, several months to many years</td>
</tr>
<tr>
<td>Other quality of service requirements</td>
<td>Real-time constraints, unobtrusiveness, stealth and others</td>
</tr>
</tbody>
</table>
3.1.1 Deployment

Due to the nature of SB-WSNs, deployment is much more sophisticated than is the case with other types of WSNs. The satellite nodes are launched using launch vehicles and released from the vehicle in a pre-calculated manner. A suitable approach for deploying SB-WSNs is that taken by the Poly Picosatellite Orbital Deployer (P-POD) project [71]. PPOD is a standardized CubeSat deployment system. It is capable of carrying three standard CubeSats and serves as the interface between the CubeSats and launch vehicle. The P-POD is an aluminum, rectangular box with a door and a spring mechanism. CubeSats slide along a series of rails during ejection into orbit. CubeSats must be compatible with the P-POD to ensure safety and success of the mission.

3.1.2 Mobility

An important aspect of the design of any WSN is the nature of the relative mobility between the nodes, the nodes and the sink or the nodes and the phenomena to be sensed. The relative mobility of the nodes is subject to the orbital geometry of the SB-WSN. The orbital geometry influences the satellite coverage and visibility of satellites. It also affects the physical propagation considerations such as power constraints and link budgets. The influence of the orbital geometry also extends to a particularly important set of factors, from a networking viewpoint: the resulting dynamic network topology and round-trip latency and variation. As a result of this, the choice of orbits and the resulting satellite network topology must be considered carefully and characterized accurately.

There are two general types of formation design configurations [72];

1. Formations that have spacecraft in the same orbital plane
2. Formations that have spacecraft in different orbital planes

An example of the first formation design configuration is the leader-follower pattern. The leader-follower configuration has satellites flying in the same orbital planes which are separated only by the mean anomaly. Satellites lying in the same orbital planes may have different eccentricities and/or altitudes. One of the examples of the second design configuration is the In-Track formation (also called same-ground formation). The In-Track formation has two or more satellites orbiting in slightly different orbital planes, which are separated by shifts in true anomaly, $\delta \theta$, and right ascension, $\delta \Omega$. The value of $\delta \Omega$ orients the orbits so that the spacecraft in the formation share the same ground track [43]. The mathematical expression for such a Formation pattern would be

$$\delta \Omega = \omega_k \cdot \delta t$$

(3-1)
where $\omega_e$ and $\delta t$ are the angular velocity of the Earth and satellite ground passes respectively. Satellites in formations can have different planes not only when they have a shift of right ascension, $\delta \Omega$, but also when they have a slight difference in their inclinations, $\delta i$. Views of both types of formation-flying configurations as they appear in space are shown in figure 3.1 which are generated using the satellite tool kit (STK) simulator [73].

![Figure 3.1: STK view of the formation flying configurations: leader-follower (left) and same ground track (right)](image)

Each of the configurations is obtained by adjusting the orbital parameters appropriately to obtain the required configuration. The patterns for the leader-follower and the same-ground track configurations are shown in the results obtained from the STK simulations presented in figures 3.2 and 3.3, respectively. It is important to note that the effect of atmospheric drag has been ignored assuming that decay due to external disturbances is being overcome using micropropulsion.

Figure 3.2 shows that the distance between the satellites in the leader-follower configuration is constant. However, in the same ground track configuration shown in figure 3.3 distances between the satellites are continuously varying in a sinusoidal manner. Each relative motion pattern is repeated twice per orbit.

From a mobility perspective, the space-based formation flying missions with a leader-follower configuration are similar in behaviour (from mobility prospective) to the static WSNs that are common in terrestrial applications.
3.1.3 Node Design Parameters

This section discusses parameters of the SB-WSN node hardware design such as cost, size and energy. Section 2.3.2 mentions some of the remote nodes (motes) that are used in the...
area of WSNs research. The idea of the mote is to have a standard hardware design that includes a processing system and a wireless transceiver and can be used for various sensing applications by plugging in a specifically designed sensor board and programming the mote with application software.

The general architecture of a mote was shown in figure 2.5. Figure 3.4 shows the general architecture of a SB-WSN node within a formation-flying mission. The figure was designed to show the similarity with the general architecture of WSN motes. A good example of a mote is the mica-2 mote [26]. An investigation was carried out to see if a similar approach could be followed in order to produce a space-based mote. Based on the literature review (summarized in section 2.2) we conclude that the development of a spacecraft mote is not feasible with the currently available commercial-off-the-shelf components. The reasons behind this are as follows:

- Firstly, there is the need for miniature propulsion systems that fit into the picosatellite nodes mass and power budgets, in order to carry out the formation keeping and collision avoidance manoeuvres.

- A second concern with regards to the space mote design is the stringent positioning requirements that formation flying impose on the GPS receiver of the motes. The inter-satellite position knowledge requirement is in the order of 1 percent of the actual separation distance [74]. This translates to 1 m relative position knowledge for a separation of 100 m between the spacecraft. Currently it is very challenging to achieve this level of accuracy using state-of-the-art COTS GPS receivers. In addition, the orbital environment imposes additional requirements on the receiver, including the relative velocities between the user and the GPS spacecraft, which is much larger than in terrestrial applications, and therefore presents a much higher Doppler shift frequency space to be searched for by the GPS signal. Without estimating the expected relative velocity and thereby narrowing the Doppler search space, it is possible that a signal lock is never acquired.

- A third concern is the need that the mote’s GPS should account for the phase differences between nonaligned antenna bore-sights that will certainly occur between the multiple spacecraft. All of these factors make the GPS receivers for such missions, complex and therefore having large mass and high power consumption. For example, the state-of-the-art GPS receiver designed for the ORION formation-flying mission has a mass of 1 Kg, and power consumption of no less than 1.4 W [75], which is far beyond the requirements of a space-based mote.
Another reason is the need for novel thermal control techniques for ultra-small satellites. Simple lumped-parameter models of silicon satellite temperature swings between fully lit and Earth-eclipsed conditions have shown that passive thermal control is possible for nearly spherical nanosatellites and micro-satellites [76]. When dimensions drop below 2 cm, the temperature extremes exceed typical electronics and battery limits. Tiny satellites, with their extremely low mass can reach the equilibrium sunlight (or eclipse) temperature within minutes. As a consequence, picosatellites and femtosatellites will require some form of thermal control.

![Node architecture of SB-WSN](image)

In the long-term, the silicon satellite concept proposed by Janson and Helvijian presents a new paradigm for the design and construction of distributed space systems [77]. Spacecraft (made mostly of silicon) that are capable of attitude and orbit control for complex space missions can be designed for mass production using adaptations of semi-conductor batch fabrication techniques. Useful silicon satellites will have dimensions of 10 cm$^3$ to 30 cm$^3$. Initial nanosatellite designs will use two types of processed wafers: a) **Wafer type I** - wafers that consist of a sparse number of electronic devices of low interconnect density, numerous micro-channels, MEMS and MOEMS; b) **Wafer type II** - wafers that take the form of Multi-Chip Modules (MCMs) containing most of the centralized signal processing, command and control electronics and the RF communications of the satellite. More on module based design for spacecraft systems is given in section 3.2.
Chapter 3: Design Space Exploration of Space-Based Wireless Sensor Networks

The main power source for SB-WSN is the solar cells that normally cover the structure of the satellite node. In order to estimate the power generated by solar cells that would cover the surface of the nodes, the following assumptions were made:

Number of sides covered by solar cells = 4
Eclipse time = 33% of the orbit time
Efficiency of the Solar cells = 26%
Solar cell area (A) = 8cm X 10cm = 0.008 m²
Average area of the nodes pointing towards the sun and receiving solar radiation = 1.4A = 1.4 X 0.008 = 0.0112 m²

The total power produced from the solar cells is calculated from the following equation [78]:

\[ P = xAE \]  

Where,
\( P \) - Total power produced from solar cell
\( x \) - solar constant = 1367 W/m²
\( A \) - Surface area of each solar cell.
\( E \) - Efficiency of each solar cell.

\[ P = (1367 \times 66/100) \times (0.0112) \times 0.26 = 2.65 \text{ Watts} \]

The outcome of the above calculations shows that the total maximum available power for the mote is 2.65 W. This value imposes strict power requirements on the subsystems on board. The power requirement of the nodes is less that the power consumed by SSTL's OBC386 on-board Computer which consumes 5 W of power [79]. This is an indication of the challenges of constructing motes with such a limited power budget of less than 3 W.

3.1.4 Heterogeneity

Similar to 'mainstream' WSNs, SB-WSNs could consist of non-identical nodes i.e. they may differ in their degree of heterogeneity. SB-WSN could consist of a mixture of nodes that have a downlink/uplink transceiver (the relatively larger nodes in the figure) for relaying the data to Earth and others that do not have this capability. Another example of heterogeneity is that some nodes in the network may have a different type of payload than the others.

Contrary to the mainstream case, heterogeneity in SB-WSNs has some consequences. The difference in mass and cross-sectional area between the satellites causes the formation to
disperse due to the effect of the phenomenon known as atmospheric drag. Atmospheric drag is the principal non-gravitational force acting on satellites in Low-Earth Orbit (LEO) [80]. Drag acts in the opposite direction to the velocity vector and removes energy from the orbit. This energy reduction causes the orbit to get smaller leading to further increases in drag. Eventually, the altitude of the orbit becomes so small the satellite re-enters the atmosphere. The equation for acceleration due to drag on a satellite is [80]:

\[ A_D = -(1/2)\rho(C_D A/m)V^2 \]  

(3-3)

Where \( \rho \) is atmospheric density, \( A \) is the node’s cross-sectional area, \( m \) is the node’s mass, \( V \) is the node’s velocity with respect to the atmosphere, and \( C_D \) is the drag coefficient \( \approx 2.2 \).

The drift caused by atmospheric drag requires compensation using the micropropulsion system onboard the satellite nodes. The difference in the fuel requirements between the identical spacecraft case and the case where differential drag exists in various formation configurations is proportional to the difference in the orbital decay rate between to the two cases.

Table 3.2 gives a clear picture of the effects of differential drag and satellite altitude on the fuel requirements of different formation patterns: leader-follower, in-track and inclination difference. The table shows two test cases for each formation pattern: one for a formation with two identical spacecraft and one with a one satellite having 5% more drag-area than the other. The units used in the table for quantifying the difference in fuel consumption is m/s which is a unit of the parameter known in rocket engineering as delta-V [80]. Delta-V is the sum of the velocity changes required throughout the space mission life.

As can be seen from table 3.2, the “inclination difference” formation pattern is the most expensive in terms of delta-V for the purpose of formation keeping. This is because having the two orbital planes at slightly different inclinations, leads the argument of the ascending nodes of each of the two orbits to precess at slightly different rates due to the secular J2 effects [72]. J2 is an important dimensionless coefficient that quantifies the effects of oblateness of the earth on orbit and is used for finding accurate orbital properties [41].
Figure 3.5: Drift between satellites due to external disturbances that affect the lifetime of the mission.
Other formation patterns require a small amount of delta-V for the purpose of formation keeping, in case the spacecraft nodes of the formation are identical. For formations involving dissimilar spacecraft, the cost of formation keeping is very high for low altitude orbits [72]. The cost of formation keeping for a formation at a 400 km altitude with a 5% drag area difference between its spacecraft is approximately 30 m/sec/year [72].

Table 3.2: Difference between Delta-V requirements for 400 km Altitude [72].

<table>
<thead>
<tr>
<th>Delta-V</th>
<th>Leader-Follower</th>
<th>In-track</th>
<th>Inclination Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Identical spacecraft</td>
<td>5% Differential Drag-area</td>
<td>Identical spacecraft</td>
</tr>
<tr>
<td>$V_{x,y}$ m/s</td>
<td>0</td>
<td>30</td>
<td>0.010</td>
</tr>
<tr>
<td>$V_z$ m/s</td>
<td>0</td>
<td>0</td>
<td>0.0015</td>
</tr>
<tr>
<td>Total, m/s/yr (approx)</td>
<td>0</td>
<td>30</td>
<td>0.0115</td>
</tr>
</tbody>
</table>

3.1.5 Communication Modality

Two common modalities exist for communications between the satellite nodes: RF and optical. Using optical communications for small satellites is faced with the hurdle of having line-of-sight communications between the transmitter and the receiver. This means that the satellites require accurate pointing hardware to establish these links, which is certainly difficult to achieve for a formation flying mission due to the mobile nature of these formations and their relatively high-speed. It is being envisaged that missions using optical communications will most likely use both RF and optical systems together [81]. The RF system will support coarse formation control. Once the formation is in place, the optical-based fine pointing system would take over.

For missions that involve miniature satellites it is not practical in terms of power and mass to include both types of communications. Therefore we conclude that for missions that involve large spacecraft platforms where mass and power are of relatively less importance; optical communications may be used to achieve higher communications performance. However, for missions involving miniature satellite platforms optical systems are not a practical approach for communications. This is because the need of mechanical pointing
structures would mean additional mass and power, besides a significant increase in platform complexity and cost.

### 3.1.6 Infrastructure

Different applications exclude, permit or require the use of a fixed infrastructure. An *infrastructure network* is a wireless network that connects to the world through an access point to a conventional Local Access Network (LAN) [17]. An alternative configuration for wireless networking is known as an *ad hoc network*. The primary characteristic of Ad hoc networks is that they do not include an access point or base station. They are formed as a result of the mutual detection of two or more mobile devices with wireless interfaces located in the same vicinity [17]. SB-WSNs are naturally ad hoc i.e., nodes can directly communicate with each other without an infrastructure. The argument that the need of certain nodes to act as downlink nodes for communications with the operators on Earth imposes an infrastructure on SB-WSNs is not valid. This is because in infrastructure networks the nodes can only communicate with the so-called basestation and the communication between the nodes are therefore relayed via the basestation. In addition, in infrastructure networks the basestations needs to have the ability to communicate with each other. This is obviously not the case in SB-WSN.

### 3.1.7 Network Topology

Network topology could be single hop, star, multihop, mesh and/or multtier. The simplest form a WSN could take is a single-hop network, where each node in the network is able to communicate with every other node in the network. On the other hand, an infrastructure-based network with a single base station forms a star network with a diameter of two [70]. A multi-hop network may form an arbitrary graph, but often an overlay network with a simpler structure is constructed such as a tree or a set of connected stars. In summary, the SB-WSN network topology depends on the diameter of the network. SB-WSN is a term that we have given to formation-flying missions that consist of ultra-small satellite nodes, which in most cases need to be deployed in large numbers in order to satisfy the motivation behind them. For this reason, the network topology of SB-WSN is multihop.

### 3.1.8 Coverage

Coverage could be sparse, dense or redundant. Coverage in SB-WSN has different dimensions than coverage in ‘mainstream’ WSN. The importance of coverage in the design of space missions lies in that it has a significant influence on the choice of orbits for the
mission. In SB-WSN the nature of the coverage depends on the payload being used to carry out the measurements required to achieve the mission objectives. Earth coverage, for example, refers to the part of the earth that the spacecraft instrument can see at one instant or over an extended period [80]. At least four key parameters for Earth coverage exist as follows [80]:

- **Footprint Area** = area that a specific instrument can see at any instant.
- **Instantaneous Access Area** = all the area that the instrument could potentially see at any instant if it were scanned through its normal range of orientations.
- **Area Coverage Rate** = rate at which the instrument or antenna is sensing or accessing new land.
- **Area Access Rate** = the rate at which new land is coming into the spacecraft’s access area.

These parameters are also applicable to other kinds of missions like space weather missions, for example.

### 3.1.9 Connectivity

Connectivity in WSN could be characterized as continuous, occasional or sporadic. Two factors determine the connectivity of a network: the communication ranges and physical locations of individual nodes [70]. This makes connectivity of SB-WSNs a characteristic of this type of WSNs. The relative mobility of the nodes makes the connectivity sporadic.

### 3.1.10 Network Size

The number of nodes participating in a SB-WSN is mainly determined by the requirements relating to network connectivity and coverage, and by the nature of the phenomenon to be sensed/studied. The network size may vary from a few nodes to thousands of sensor nodes and even more. The network size determines the scalability requirements with regard to protocols and algorithms.

### 3.1.11 Lifetime

A WSN lifetime is the time period from the deployment of the network to the instant when the network is considered nonfunctional [82]. The definition of the instant when a network becomes nonfunctional is application-specific. It can be, for example, the instant when the
first sensor dies, a percentage of sensors die, the network partitions, or the loss of coverage occurs.

Usually, the main cause of a network becoming non-functional is the depletion of the battery of the nodes. In SB-WSNs, the nodes usually have solar panels that enable them to recharge their batteries. However, there exist other reasons for nodes exiting from the SB-WSN. If the fuel that a node carries is completely consumed, the nodes start losing altitude due to drag until it diverges away from the network’s coverage area. More on the influence of fuel on the lifetime of SB-WSNs was discussed in section 3.1.4.

3.1.12 Quality of Service requirements

An important aspect of SB-WSNs is that the system has real-time constraints, mainly due to the presence of a control loop in order to maintain the formation.

Another design choice to be made for SB-WSNs is the organizational architecture of the network. Three organisational architectures for formation flying missions are proposed in [83] as follows:

- **Master/slave**: One spacecraft acts as master while the others are slaves. The slaves transmit sensor values to the master that performs all the autonomy reasoning based on the data it receives. The slave spacecrafts forward control signals they receive from the master to the appropriate local devices including subsystems, sensors and actuators. The basic advantage of the master/slave approach is its simplicity. The disadvantage is that it relies on the assumption that the master spacecraft’s reactive controller can continuously monitor the slave’s hardware, and this relies on high-bandwidth highly reliable communications.

- **Teamwork**: The teamwork approach uses a leader/follower configuration in which plans are constructed centrally at the leader and the actions are broadcast to each follower.

- **Peer-To-Peer coordination**: Here each spacecraft is able to function as both a leader and a follower. The planning process is distributed among the spacecraft in the constellation.
3.2 Module-Based Methodology to Designing SB-WSN

A novel methodology for the design of SB-WSNs is proposed in this section. The methodology is based on the availability of standard hardware and software components that enables 'plug and play' space missions. The approach relies on the assembly of standalone software and hardware modules that can be interconnected to construct SB-WSN nodes. The advantage of the module based approach over the single module spacecraft design approach conventionally used for space missions is that it provides a suitable trade-off between flexibility and low-cost.

The overall design flow is shown in figure 3.6. The requirements of the mission are used to identify the mission concept in terms of the general mission design factors, particularly the factors that affect the mission in general such as orbit design, inter-satellite link design and payload design. The mission concept phase is used to identify the budgets for the individual spacecraft nodes. The module architecture design phase is where the node design is partitioned into modules. In case no modules exist with the identified requirements the MCM design phase is used to design the architecture of the module. The MCM design phase is followed by the design of the System-On-Chip (SoC) which involves the FPGA hardware design process. The software modules are then designed followed by the phase in which the system integration takes place.

The effectiveness of the design flow is difficult to verify, without completing the entire process of mission design including the different aspects of formation flying, inter-satellite links and the design of the spacecraft nodes including the MCMs involved. This is obviously beyond the scope of this thesis. However, the modular nature of the methodology makes its advantages obvious, given that certain standards are provided in order to preserve the modularity advantages.

The possible standards to be set for the modules of the satellite nodes are related to: data interface, size and electrical characteristics. The size and electrical characteristics depend very much on the nature of the module itself and its functionality. For example a propulsion module containing mechanical components would require much higher power consumption that an OBC module. The most important standard that needs to be defined is the data interface.
Several data interface standards are currently in use in the space industry. Very popular on-board buses for small satellites are MIL-Std-1553 [84] [85], CAN [86] and SpaceWire [87] [88]. A comparison of these interfaces is shown in table 3.3. From table 3.3 we can categorise the on-board interfaces into two sets: a low-data rate set and a high-data rate set. This is because these are not comparable with each other, they are supplementary i.e. low data-rate interfaces have different applications than the high data-rate interfaces. High-rate applications include those that involve the transfer of imaging (and other payload) data. Low data-rate applications include those that involve the transfer of spacecraft housekeeping data.
Table 3.3: Comparison between the different space qualified interfaces.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Data-rate</th>
<th>Power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIL-STD-1553</td>
<td>1.0 Mbps</td>
<td>1.475, 1.46, 1.79 W (depending on the radiation tolerance)</td>
</tr>
<tr>
<td>[84] [85]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN-bus [86]</td>
<td>1.0 Mbps</td>
<td>0.3 W</td>
</tr>
<tr>
<td>SpaceWire [87] [88]</td>
<td>400 Mbps</td>
<td>12.5 mW</td>
</tr>
<tr>
<td>Zigbee [89]</td>
<td>250 Kbps</td>
<td>&lt;1 mW</td>
</tr>
</tbody>
</table>

For high data-rate onboard applications SpaceWire [87] [88] is selected as an interface for the modular architecture. The reason behind this choice is that SpaceWire offers a good combination of scalability and low-power. For low data-rate applications Zigbee was found as a promising wireless interface offering ultra-low power consumption [89]. Zigbee (IEEE 802.15.4) is a wireless protocol designed for low-power short range data transmission. Zigbee is of interest for SB-WSN applications mainly due to the ultra-low power consumption that it offers. While Zigbee is suitable for applications that involve certain sensor data handling, it is not suitable for all of the on-board applications (e.g. imaging data transfer between mass memory and on-board computer modules) which require high data-rates. SB-WSNs are expected to have significant computational power distributed within the node; which makes a high-speed data link necessary. For this reason, the on-board interface used for the purpose of this thesis is limited to the wired SpaceWire interface.

SpaceWire was designed specifically for space applications. It is a point-to-point serial link that is composed of nodes that are interconnected via routing switches [87] [88]. In this chapter, we have only defined the SpaceWire interface for usage as a standard data interface for the modular architecture. More on SpaceWire and its implementation will be presented in Chapter 4 in the context of the implementation of the On-Board Computing module.

3.2.1 MCM Design Flow

The MCM design phase of the module design involves the physical design of the 3D-MCM chip as decided in the previous design phase (the module architecture design phase). The MCM design flow followed by 3D-plus [90] is explained in the flowchart of figure 3.7.
The mechanical development of a 3D module is very similar to that used for standard 2D MCM. The main difference is in the splitting of the design function into several layers.

Each of these layers is essentially an individual circuit board that allow the stacking of electronic components placed on them. Once each layer (circuit board) of the module is designed, it can be fabricated, populated with components, tested, and burned in, all before final module assembly. The layers are then stacked together, aligned, and vertically spaced.

A very pure epoxy resin from Dexter (HYSOL FP4450) is used to fill the spaces around the entire module and between each layer. After resin polymerization, the module is cut from the mold, exposing the external connections (flying-leads) that will be used to vertically interconnect the layers. Nickel is then chemically and electro-chemically deposited onto the module. This effectively shorts all of the flying leads together. A laser is then used to create grooves that leave the appropriate vertical connections. Finally, a mission specific thickness of Tantalum is installed for radiation shielding [91].

3.2.2 System-On-Chip Design Flow

The System-on-chip design is aimed at the generation of the hardware design that is downloaded in to the FPGA within the MCM. The system-on-chip design includes two basic important Intellectual Property (IP) cores on which the basic SoC design is based on: the processor and the SpaceWire interface.

After the architectural design of the SoC, the implementation follows the traditional flow that is shown in figure 3.8. ModelSim is a well-engineered logic simulator from Model Technology for hardware designs written in VHDL, Verilog or SystemC or mixture of these three languages. It compiles the sources and simulates it. With 9 different views (wave diagram, dataflow, signals, variables, etc.) a design can be analysed, simulated and debugged easily.
### Electrical Design
- Component selection
- Design Capture and Verification
- Electrical analysis (Power dissipated – Grounding allocation, Critical signals paths, EMC, Input capacitances)

### Structural Design:
- Thermo-mechanical analysis
- Thermal analysis: heat dissipation modelling within a 3-D module in conduction, convection and radiation cooling modes.

### 3-D Module final design:
- Product definition data file
- Associated technical and quality documentation

### Manufacturing

### Testing
- Test software and hardware development
- Electrical characterization
- Device programming
- Radiation test for Space applications (Total Dose, SEU, SEL)
- Module screening (MIL-STD-883 or ESA-PSS)

### Custom Evaluation and Qualification capabilities:
- Reliability testing/qualification
- Thermal shock, thermal cycles
- Vibration (sinusoidal and random)
- Temperature & humidity

---

**Figure 3.7: 3D-Plus MCM Design flow [90]**

Xilinx ISE is a software environment with many tools, which provides all steps of the hardware design flow from editing the sources via synthesis up to downloading the design into a Xilinx FPGA. Xilinx ISE 5.1 has been used during this project. The synthesis tools of Xilinx were not able to synthesise the used LEON processor core. Therefore the Synplify tool has been used during the synthesis step.
3.2.3 Software Design Flow

The design methodology for the software development depends on the software platform that is adopted for the system. The choice of the operating system (which is also dependant of the chosen software model chosen) also affects the design flow of the software. Hence before presenting the software design flow, this subsection first presents the selection procedure for the operating system, which is an important part of the software platform.

3.2.3.1 Operating System Analysis

There is hardly any research project specifically targeting operating systems design for space missions. The majority of modern space missions use Commercial Off-The-Shelf (COTS) operating systems that are usually aimed at the terrestrial embedded market such as VxWorks and QNX. QNX have been flown on various Surrey satellites including UoSat-12 [93]. QNX has the advantage of being a microkernel. A microkernel is a minimal operating system that offers only the necessary mechanisms for implementing operating system services in user mode [94]. FlightLinux is an interesting project with the goal of providing and on-orbit flight demonstration of the Linux operating system [95]. The main motivation behind Linux is the reduction of software development costs. The disadvantage of Linux is that it lacks real-time performance, which is a requirement in spacecraft software design. RT-Linux is a real-time version of the mainstream Linux but with some serious
disadvantages: has a relatively large footprint (due to having two kernels) and is an order of magnitude slower than typical RTOSs [96].

For the purpose of our research, we decided to select two representative operating systems from two categories of the operating systems that are shown in table 3.4. The selected operating systems would be compared quantitatively and qualitatively. The following criteria were put for the initial selection.

- Embedded operating system: The selected operating system should be designed specifically for embedded platforms

- Open-Source: This is mainly for three reasons. Firstly to have access to the source code, secondly in order to overcome the limited funds problem.

The initial selection resulted in choosing two operating systems for comparison: RTEMS and TinyOS. A general comparison between TinyOS and RTEMS is given in table 3.5. RTEMS is a real-time executive that has several advantages for being used on-board space missions. One of those advantages is that it supports real-time applications. Space examples of hard real-time applications are those of attitude control, spacecraft clock maintenance, and telemetry formatting. Examples of soft real-time applications include thermal control and bulk memory scrubbing. Memory scrubbing is the process of retrieving data from the memory and performing checks on it in order to detect and correct possible errors [97].

One disadvantage of RTEMS is that it does not allow dynamic task loading as applications are actually wrapped by the operating system and then uploaded to the satellite. An advantage of RTEMS is its support for multi-processor system design and that it naturally supports distributed system implementations.

A completely different approach to the RTEMS operating system is the approach taken in the design of the TinyOS operating system (that have been previously briefly mentioned in chapter 2) which seems to be an inspiring approach for the design of operating systems for SB-WSNs. TinyOS [14] is perhaps one of the earliest operating system kernels developed exclusively for sensor nodes. The paradigm shift in distributed computing brought about by the advent of sensor networks required revisiting the basic operating system abstractions such as tasks and inter-task communication, as well as developing support for fundamentally new distributed programming environments [98]. In addition, the severe resource limitations, reliability considerations, real-time constraints, and unpredictability of the environment called for creative implementations of basic kernel functions.
Table 3.4: Comparison between the different types of operating systems

<table>
<thead>
<tr>
<th>OPERATING SYSTEM</th>
<th>Description</th>
<th>Pros</th>
<th>Cons</th>
<th>Example/ Mission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic</td>
<td>Almost any procedure can call any other procedure.</td>
<td>Efficient</td>
<td>Lack modularity</td>
<td>OS: Linux</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mission: None</td>
</tr>
<tr>
<td>Microkernel (client/server)</td>
<td>A few essential functions are embedded in the kernel. Other services run as processes in user mode.</td>
<td>Flexible</td>
<td>Less efficient than monolithic</td>
<td>OS: QNX, VxWorks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Well suited for distributed systems</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Missions: TiungSAT-1, PROBA</td>
</tr>
<tr>
<td>Virtual Machines</td>
<td>Exact copy of bare hardware.</td>
<td>Portable</td>
<td>Low-performance</td>
<td>OS: Embedded</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Java Virtual</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td>Component-Based</td>
<td>The Operating system consists of a set of independent components representing system resources</td>
<td>Portable, Efficient, Well suited for distributed systems</td>
<td></td>
<td>OS: TinyOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mission: None</td>
</tr>
</tbody>
</table>

TinyOS is a component-based, event-driven operating system that provides support for communication, multitasking, and code modularity. The TinyOS framework is designed to have a tiny footprint starting at a few hundred bytes for the scheduler and grows to complete network applications in a few kilobytes [98]. TinyOS adopts an event model as to provide the ability to handle high levels of concurrency in a very small amount of space. The TinyOS design library provides basic system services such as communication and simple process scheduling, and access to hardware components, such as sensors and actuators.

TinyOS uses a lightweight approach that is different from traditional monolithic operating systems, where the operating system is downloaded to the system as a full-sized binary. The structure of TinyOS itself is made up of a set of components. These components are assembled altogether with the application by the user. The advantage of this approach is that only the component whose functionality is required by the application is included in the image which is finally installed in the memory of the embedded system. Another advantage of this approach is that the user applications can be made up of several collaborating components and thus enforcing reuse [14].

45
The TinyOS communication model is based on the Active Messages paradigm [14] [99]. According to it, the structure of the messages exchanged between the nodes contains the ID of a handle to be invoked on the target node and data payload to pass in as arguments. This event-based and message-oriented communication paradigm makes TinyOS a good foundation for building a publish/subscribe-based architecture.

A significant disadvantage of TinyOS is that it is not currently a real-time operating system. The main problem with TinyOS is that it is not a real-time operating system. However, it is possible to adapt the operating system in order to enhance its capabilities with real-time performance [100]. Another disadvantage of TinyOS is that it lacks memory protection [101]. However, a memory safe version of TinyOS was reported in [101].

TinyOS is written in NesC [102] using a component-based architecture with layered access to hardware resources, which provides robustness, flexibility, and extensibility. NesC is explained in more details in section 6.2 in the context of middleware design.

Table 3.5: Comparison between RTEMS and TinyOS embedded operating systems

<table>
<thead>
<tr>
<th></th>
<th>RTEMS</th>
<th>TinyOS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model Of Computing</strong></td>
<td>Multi-Tasking</td>
<td>Communicating EFSMs</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>Free (open-source)</td>
<td>Free (open-source)</td>
</tr>
<tr>
<td><strong>Scheduling</strong></td>
<td>SCHED_RR, SCHED_FIFO</td>
<td>FIFO, preemptive</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td>pThreads (Posix Threads)</td>
<td>Limited- Two Threads of Execution: Tasks and hardware event handler</td>
</tr>
<tr>
<td><strong>IPC</strong></td>
<td>Semaphores, Mutexes,</td>
<td>Exclusive shared memory</td>
</tr>
<tr>
<td></td>
<td>condition-variables, Pqueues (Posix Queues)</td>
<td></td>
</tr>
<tr>
<td><strong>Footprint</strong></td>
<td>64K-128K</td>
<td>400bytes</td>
</tr>
<tr>
<td><strong>Debug</strong></td>
<td>GDB, DDD</td>
<td>TOSSIM (Simulator + debugger) [103]</td>
</tr>
<tr>
<td><strong>API</strong></td>
<td>POSIX, RTEMS API, uITRON</td>
<td>Custom</td>
</tr>
</tbody>
</table>

46
3.2.3.2 Software Design Flow

Figure 3.9 shows the development process of sensor node software. First, for each functional block the components must be identified and included. During the design phase, the chosen components from the operating system and the middleware are interconnected and dependencies are resolved. During this phase, the interface as well as the parameter optimization is done and the final source code generated. Additionally, the runtime behaviour can be monitored by including the logging components. Next, during the compilation process, the executable is created. The compilation process flowchart is shown in figure 3.10.

The application program (along with the middleware components) which is in nesC is compiled using a nesC compiler that converts the program to a single C file. This C file is then passed through the normal embedded systems design flow. The C file is compiled (using the GCC compiler), and an executable file is produced.

![Software design flow for the nodes](image)

The final step of the design flow, the evaluation phase, the created node application can be downloaded to the node and executed. Considering the monitoring results an improved design cycle can be started. As a result of the design flow, optimized node application
software is produced. The node application consists of special tailored parts only needed by the specific application of the node.

![Diagram showing the compilation of application programs for node software](image)

Figure 3.10: Compilation of the application programs for the node software [104]

### 3.3 Conclusions

This chapter covered the design space of SB-WSN. The different aspects that affect the design of SB-WSN have been studied. The dimensions of the design space include deployment, mobility, the node’s design including cost, size and energy considerations, heterogeneity, communication modality, infrastructure, network topology, coverage, connectivity, network size and lifetime. The most important outcome of this study is the identification of the differences and similarities between mainstream WSNs and SB-WSN.

These similarities can be summarized in the following points:

- The general architecture of the network consisting of a data collecting nodes interconnected via a wireless network and sending the data towards a sink that needs the data.
- Highly-constrained environment
- The need for self-configuration
- The need for scalability

The study also reveals the differences between SB-WSN and WSNs:

- Mobility. The nodes in SB-WSN are mobile and their mobility is predictable. This is a unique feature of SB-WSN.
- Complexity (and hence the cost) of the nodes. Satellite nodes maybe considered as multi-sensor, multi-actuator motes.
The extremity of the environment. The space environment incurs additional requirements on the design of the nodes due to factors like radiations, thermal conditions etc.

The need of complex mechanisms for deployment in the case of SB-WSN which is not the case for most WSN applications.

The lifetime for SB-WSN is not dependent on the power available in the node; it is dependent primarily on the fuel remaining.

The distances between the nodes. In case of SB-WSN the separation distances is much greater than with other types of WSNs. This has significance mainly on the design of communications data link design.

The second section of the chapter proposed a design methodology for SB-WSN. The design is based on a modular architecture in which the nodes themselves are considered as a collection of hardware and software modules. The motivation behind this approach is to simplify and reduce the costs associated with the design of SB-WSN. The module-based methodology offers an appropriate trade-off between flexibility and miniaturization that is required for SB-WSN node design. This is contrary to the approaches that have been highlighted in section 2.4, with the SSTL stack approach offering flexibility at the expense of miniaturization and the ChipSat approach offering miniaturization at the expense of flexibility.

The power behind the module-based system lies in the ability to simplify interfacing the components together. This requires a standard set of interfaces to connect the components together. For the purpose of the hardware modules, SpaceWire was found to be the most suitable COTS interface available for SB-WSN motes. This is because of the need of a combination of high data-rate, low-power and simplicity requirements for the interface. Zigbee is also a suitable wireless alternative to SpaceWire, for missions that require low data-rates for their nodes.
Chapter 4

4 Case-Study: Design of an On-Board Computer Module

This chapter provides the details of an example design of a miniaturized hardware module using the methodology proposed in chapter 3, which involves the integration of several standalone hardware and software modules in order to meet the mission specification. This module corresponds to the on-board computer (OBC) of a modern micro-satellite. The first section of the chapter reviews the literature on on-board computer miniaturization. Section 4.2 gives details about the design of the proposed on-board computer module. Section 4.3 presents implementation results.

4.1 Review of On-Board Computer Miniaturization

The onboard data-handling system (OBDH) is an important part of a small satellite. The OBDH system is the key to the sophisticated capability of microsatellites [67]. Early OBDH systems functionality was limited to command and telemetry. However, typical functions of a modern OBDH system also include controlling payloads and subsystems, enabling the flow of housekeeping and science data, receiving and distributing commands, performing telemetry and tele-command protocols, time distribution around the spacecraft, providing data storage, executing commands and schedules, monitoring spacecraft health and performing data compression [105]. Future OBDH system functions will also include the implementation of artificial intelligence components for autonomy purposes.

The OBC can be described as the kernel of the on-board data handling system. In this section we investigate state-of-the-art COTS OBCs by comparing representative computers designed for on-board use. These OBCs are: the RAD-750 OBC [106], the RH-PPC OBC [107] and the SSTL OBC386 [79]. The RAD-750 is the platform that was going to be used on the Techsat-21 mission [108]. A comparison between the boards is given in Table 4.1. Although all of the three OBCs mentioned above demonstrate high-performance, they remain massive and power hungry if looked at from the prospective of SB-WSNs. From table 4.1 it is seen that the weights of the OBCs are 540grams, 1.12Kg and 1.7Kg. The
Chapter 4: Case-Study: Design of an On-Board Computer Module

OBC with the lowest mass, the RAD750, has a mass of 540 grams which is more than 50% of the weight of a picosatellite (1 Kg). With regards to power consumption, the OBC386 has the lowest power consumption (5W). In section 3.1.3, the total power available to a 10 cm³ picosatellite is in the range of 2.84 Watts. The OBC386 therefore has a power consumption that is nearly twice the total power consumption available for the complete SB-WSN node. There is obviously a need of miniature low-power OBCs with comparable performance to the state-of-the-art OBCs.

Table 4.1 State-Of-The-Art On-Board Computers

<table>
<thead>
<tr>
<th>Single Board Computer</th>
<th>RAD 750</th>
<th>RHPPC</th>
<th>OBC386</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Memory</td>
<td>128MB SDRAM</td>
<td>128MB (DRAM)</td>
<td>64-256MB</td>
</tr>
<tr>
<td>Processor</td>
<td>RAD (PowerPC) 750 240MIPS 132MHz</td>
<td>PowerPC 603e 210MIPS 150MHz</td>
<td>386 EX 16-25MHz 5MIPS</td>
</tr>
<tr>
<td>Non-volatile Memory</td>
<td>256KB SUROM</td>
<td>4MB EEPROM, 128KB SUROM</td>
<td>32KB PROM</td>
</tr>
<tr>
<td>Serial I/O</td>
<td>UART</td>
<td>1553/1773 2 full duplex UART 8250</td>
<td>2 CAN connections 10Mbps Ethernet</td>
</tr>
<tr>
<td>DMA Controllers</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Power management</td>
<td>Multiple power down mode</td>
<td>Multiple power down modes and frequencies</td>
<td>None</td>
</tr>
<tr>
<td>Backplane Bus</td>
<td>cPCI</td>
<td>Dual cPCI</td>
<td>-</td>
</tr>
<tr>
<td>Operating System</td>
<td>VxWorks 5.4</td>
<td>VxWorks</td>
<td>POSIX or SCOS</td>
</tr>
<tr>
<td>Power</td>
<td>12 Watts</td>
<td>12.5 Watts</td>
<td>5 Watts</td>
</tr>
<tr>
<td>Weight</td>
<td>540 grams</td>
<td>1.12 Kg</td>
<td>1.7 Kg</td>
</tr>
</tbody>
</table>
Several projects were reported in the literature with the aim of miniaturizing of the OBC. Researchers at NASA's Jet Propulsion Laboratory (JPL) have completed the development of a single package multichip module (MCM) flight computer. It weighs 89 grams, has a volume of less than 1.5 cubic inches, and can be used to perform the basic spacecraft on-board data handling and processing functionality [109]. The plan of the proposed program was to develop a total of three MCMs for the following building block components: a Flight Computer MCM, a Mass Memory MCM and a Programmable I/O MCM. The flight computer contains the following components: 1 CPU, 2 MMUs (caches), 4 FPGAs, 6 EEPROMs, and 20 SRAMs (128Kx8). A total of 33 bare die are placed inside the single MCM package which is approximately 2 inches by 4 inches in size. This is an old project (1995) in which the components used are out-dated.

A more recent project is that undertaken by the Surrey Space Centre (SSC), which aims at integrating a complete OBC on a single chip [69]. A number of peripheral cores have been developed such as DMA controller [110], CORDIC coprocessor [111], HDLC controller (developed by Surrey Satellite Technology Limited (SSTL)), Error Detection and Correction (EDAC) Core (from SSTL), a bootstrap loader, a fault-tolerant Advanced Encryption Standard (AES) core [112] and a lossless compression core.

A relevant extension of SSC's OBC is the work done on the reconfigurability aspects of the OBC in orbit [113]. Reconfigurability allows partial run-time reconfiguration of the SoC OBC while operating in orbit using a Xilinx Virtex FPGA. The motivation behind enabling hardware reconfiguration of the OBC is to repair and debug hardware faults that are caused due to the harsh environment that the spacecraft operates in. In addition, reconfiguration could enable hardware changes aimed at upgrade or repair of on-board electronic components and a change of functionality in response to changed mission requirements.

Another similar project is the Spacecraft-Controller-On-a-Chip (SCOC). The European Space Agency (ESA) have contracted Astrium to develop the SCOC that includes most of the peripherals along with the controller on a single chip, keeping in mind the Failure, Detection, Isolation and recovery (FDIR) scheme in implementation [114]. Since an ASIC foundry of such size is very expensive, a first step using large FPGA for the prototyping of SoC was made. The SoC developed comprises the following functions: LEON SPARC CPU with FPU, Backplane bus controller (PCI), Spacewire Bus interfaces, CCSDS packet telecommand and telemetry system, telemetry housekeeping generation, 1553 system bus controller/bus monitor/remote terminal and a time management system. The SoC is implemented on a purpose-built demonstration board, and the ASIC is integrated in a Xilinx Virtex-E FPGA.
4.2 Module Design

The design of the on-board computer module necessitates the consideration of two main requirements: miniaturization and flexibility. It is proposed that miniaturization is achieved through the use of MCM technology. The flexibility is attained by including a reconfigurable logic chip as the main component within the MCM. A system-on-chip that includes the main functionality of the module can be then designed and downloaded to the reconfigurable chip. This section is divided into two subsections: the MCM design and the design of the SoC.

4.2.1 Multi-Chip Module design

This subsection describes the design of the on-board computer module. The on-board computer module is proposed to be designed as a 3-D module using the technology that 3D-Plus [115] provides.

The proposed module needs to include the following components:

- An SRAM Field Programmable Array, to provide the core of the required functionality to the module. The OBC module exceptionally would require a high-density FPGA as it needs to include a powerful processing element that requires a relatively large number of gates.

- 4-Mbit In-System programmable configuration PROM, to store the FPGA configuration. 4Mbits are shown in section 4.3 to be enough to hold the SoC designed in the next section.

- 3-Gbit SDRAM @100MHz (6 x 64M x 8bit), to serve as memory of the on-board computer module.

- Power-On-Reset.

- 1.5-V Fixed-Output LDO Voltage Regulator for powering the FPGA core logic (VCCINT)

- Dedicated Current-limited, P-Channel Switches with Thermal Shutdown (3.3-V/500mA and 3.3-V/1.5A)

- Pull-up Resistors

- Decoupling Capacitors
The first important selection to be made, after choosing the design technology is to select the FPGA. FPGAs can usually be classified on the basis of the programming technology that they use. There are three competing technologies for programming FPGAs: SRAM-based, antifuse-based and flash-based. SRAM programming involves a small static RAM bit for each programming element. Writing the bit with a zero turns off a switch, while writing with a one turns on a switch. Antifuse FPGAs consists of a microscopic structure that, unlike a regular fuse, normally makes no connection. A large amount of current during programming of the device causes the two sides of the antifuse to connect. A third and relatively new method uses flash EPROM bits for each programming element.

The option of using an antifuse FPGA is ruled out; as these devices lack the flexibility option that is required for the OBC module. Both SRAM-based FPGAs and Flash-based FPGAs fulfil the flexibility requirement as both are reconfigurable. Flash-based FPGAs have several advantages over SRAM-based FPGAs of which the most important are that they are low-power, and are naturally immune to Single Event Upsets (SEU). There are two disadvantages of flash-based FPGAs when compared with SRAM-based FPGAs: they do not allow run-time reconfiguration and they have limited capacity compared to the SRAM-based FPGAs (the Actel ASICpro has a capacity of up to 1,000,000 system gates [116]). SRAM-based FPGAs are reconfigurable and have relatively large capacities compared to their flash-based counterparts. However the disadvantage of SRAM FPGAs is that they are slower and more power consuming than anti-fuse FPGAs.

An important aspect that was taken into account while selecting the FPGA device for the OBC module is the resistance to radiation effects that the device could be exposed to in space. There are three effects of radiation that needs to be considered [67]: Total Ionizing Dose, Single Event Latch-up and Single Event Upset. Total Ionizing Dose (TID) damage is due to the cumulative effect of ionizing radiation over time. The result is a catastrophic device failure. A device’s tolerance to total dose is measured in krads. Satellites typically encounter TID between 10 krad and 100 krad [117]. The TID of the Xilinx QPro Virtex and Virtex II FPGAs are 100 krad and 200 krad respectively, which indicates that they are very tolerant to TID. Actel Corporation has evaluated their ProASIC flash-based programmable FPGA family. The results presented in [118] conclude a TID measurement of about 60 Krads.

Single Event Latch-up (SEL) is another catastrophic condition and is caused by a single energetic ion initiating a runway current flow in the device leading to failure [67]. Normally SEL is measured by LET which is a measure of the energy deposited per unit length as an energetic particle travels through a material [118]. The SEL immunity of the Qpro Virtex-II can reach to an LET of 160 MeV.cm²/mg indicating that they are SEL immune. Results
presented in [118] conclude that for the ProAsic flash-based FPGA family n SEL occurred at an LET of 37.4 MeV.cm²/mg.

Another main class of radiation effects is the Single Event Upset (SEU) which is a change of state or transient induced by an energetic particle such as a cosmic ray or proton in device. Flash-based FPGAs are less vulnerable to SEUs than their SRAM counterparts [120]. However, quoting T. Speers et al. [119]:

"The SRAM FPGA may be superior for in-orbit reconfiguration applications, depending on the severity of the radiation effects observed during the programming of the Flash FPGA"

While both the flash-based FPGAs (the Actel ASICpro in particular) and the SRAM-based FPGAs are suitable for the purpose of the OBC Module, it was decided to use SRAM technology. The motivation behind this decision is because of three reasons. Firstly, the relatively much higher capacity of SRAM-based FPGAs. Xilinx produces SRAM FPGAs that have high capacities (up to 6 million system gates for the radiation hardened Qpro Virtex-II) [121]. Secondly, future work on the OBC module would include run-time reconfiguration of the FPGA. The third reason for using SRAM-based FPGAs for the OBC Module is the availability of radiation-hard SRAM-FPGAs.

Table 4.2 compares the Xilinx FPGAs. It is clear from the table that the XC2V8000 device is the largest FPGA in terms of capacity whereas the largest radiation tolerant FPGA is the XQ2V6000. However the lightest package of the XC2V8000 has a mass of 14.24 grams whereas the XC2V3000 (and its radiation hard equivalent – XQ2V3000) has a mass of 3.3 grams. It will be shown in the next section that the XC2V3000 is good enough for the purpose of the OBC module.

<table>
<thead>
<tr>
<th></th>
<th>XCV800</th>
<th>XQVR1000</th>
<th>XC2V3000</th>
<th>XQ2V6000</th>
<th>XC2V8000</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Gates</td>
<td>800,000</td>
<td>1,124,022</td>
<td>3,000,000</td>
<td>6,000,000</td>
<td>8,000,000</td>
</tr>
<tr>
<td>Slices</td>
<td>9,408</td>
<td>15,552</td>
<td>14,336</td>
<td>33,792</td>
<td>46,592</td>
</tr>
<tr>
<td>CLB Array</td>
<td>56 X 84</td>
<td>64 X 96</td>
<td>64 X 56</td>
<td>96 X 88</td>
<td>112 X 104</td>
</tr>
<tr>
<td>BlockRAM</td>
<td>112 Kbits</td>
<td>128 Kbits</td>
<td>1728 Kbits</td>
<td>2592 Kbits</td>
<td>3024 Kbits</td>
</tr>
</tbody>
</table>
The design flow of the MCM module was outlined in section 3.2.1. The outcome of the design flow is a 3D-MCM that incorporates all of the hardware components of the design. An illustrative picture of a 3D-MCM module designed by 3D-Plus [115] is shown in figure 4.1.

![Figure 4.1: A picture of an MCM Module from 3D-Plus[115]](image)

The cost for the development of a 3D module with the above specification by 3D-Plus is in the order of 100,000s Euros (~100,000 Euro). The manufactured part would cost around a 5,000 Euros [90]. The costs of the module design are shown in table 4.3. Due to the high costs, the implementation of the module has not been attempted. However an FPGA design has been undertaken which can serve as a proof-of-concept as would be shown later in this section. The high cost associated with the development is the motivation behind incorporating reconfigurability as part of the requirements of the OBC Module design. This is because modules can be developed initially at a high cost and then the manufactured part can be reused at a low cost through altering the functionality of the module using its reconfigurable features.

Figure 4.2 shows the mechanical design of a MCM module that is designed by 3d-plus that includes very similar components to the ones proposed for the OBC Module. The MCM is called the Radiation Tolerant Intelligent Memory Stack (RTIMS) and incorporates a XC2V1000 FPGA, 3 Gbits of SDRAM, a linear power regulator and a 4 Mbits PROM; all stacked into a module of 42.5 mm x 42.5 mm x 12.7 mm [122]. This MCM is mechanically and functionally very similar to the proposed MCM. The power consumption of the module is 3.7 Watts.
Chapter 4: Case-Study: Design of an On-Board Computer Module

Table 4.3: Costs associated with MCM design.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Rough Order of Magnitude (ROM) Cost (Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>100,000</td>
</tr>
<tr>
<td>Module LAT (Active part)</td>
<td>10,000</td>
</tr>
<tr>
<td>Component LAT (passive part)</td>
<td>10,000</td>
</tr>
<tr>
<td>Thermal Analysis</td>
<td>50,000</td>
</tr>
<tr>
<td>RVT (Reusable Rocket Vehicle) test</td>
<td>10,000</td>
</tr>
<tr>
<td>SEE (Single Event Effects) test</td>
<td>50,000</td>
</tr>
</tbody>
</table>

Figure 4.2: Mechanical design of the OBC MCM module [122]

A prototyping board was used to emulate the functionality of the MCM described above. The prototyping board is the GR-PCI-XC2V from Pender electronics [123] that comprises an XC2V3000 FPGA, 12 Mbit PROM, 512 MB SDRAM, 8Mbit SRAM and 64 Mbit flash memory (figure 4.3). The board operates in the same way that the MCM was featured to operate. The FPGA configuration bit streams are stored in the configuration PROMs. On power-up the bit streams are loaded from the PROMS to the FPGA. The flash memory acts as non-volatile memory storing the programs to be run by the SoC. The programs are loaded to the RAM that is available onboard.
4.2.2 System-On-Chip Design for the OBC Module

The SoC design for the OBC module has two important aspects: the choice of the IP cores that constitute the SoC and the interfaces used to interconnect the cores together. The choice of the IP cores involves first of all the selection of the soft CPU, which is the key to the design of the OBC module forming the backbone of the SoC.

Three processors were analyzed and compared for the purpose of the SoC design: LEON-2 [124], Open-RISC [125] and MicroBlaze [126]. The comparison is shown in table 4.4. From the available choice of processors, the LEON processor was found to be the best candidate for use in the proposed SoC design. The reason behind this is manifold:

- It was originally designed for space applications by ESA. The processor is widely used in current ESA projects and is selected as the main CPU for high-performance onboard computers. A fault-tolerant version of the processor is available.
- It is based on a standard SPARC architecture unlike the other processors that possess a proprietary architecture (such as the MicroBlaze).
- It is an open-source core and therefore is a budget solution.
- Experience at the SSC with the LEON processor as it was used in some past projects.
Table 4.4: Comparison between available soft core processors

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>5-stage</td>
<td>5-stage</td>
<td>3-stage</td>
</tr>
<tr>
<td>License</td>
<td>GNU LGPL</td>
<td>GNU LGPL</td>
<td>Xilinx IP core</td>
</tr>
<tr>
<td>ISA</td>
<td>SPARC V8</td>
<td>ORBIS32</td>
<td>MicroBlaze ISA</td>
</tr>
<tr>
<td>Register file</td>
<td>Windowed</td>
<td>Flat</td>
<td>Flat</td>
</tr>
</tbody>
</table>

The LEON processor is a 32-bit synthesizable processor core, based on the SPARC V8 architecture [127]. The core is highly configurable, and particularly suitable for system-on-a-chip (SOC) designs. Several versions of the LEON processor have been developed. The LEON2 processor was designed under contract from the European Space Agency, available as a radiation-hardened component from Atmel [124]. The processor is also supported by a full set of software development tools, including the LEON Cross Compilation System (LECCS) and the LEON/SPARC simulator – TSIM. LECCS allows cross-compilation of C and C++ application for the processor.

The block diagram of the SoC and its interfaces with the external peripherals on the prototyping board (emulating the OBC module) is shown in figure 4.4. Two on-chip buses are provided with the LEON-2 CPU: AMBA AHB and AMBA APB. The APB is used to access peripherals and on-chip registers, while the AHB is used for high-speed data transfers. The full AHB/APB standard is implemented [128]. AHB is designed for high-performance, high-clock-frequency system modules. It acts as a high-performance system backbone bus. This bus supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral functions. In our configuration, two masters are attached onto the bus: the processor and the UART of debug communication link, and four slaves are provided: memory controller, debug support unit, JTAG, and AHB/APB bridge.

The AHB/APB Bridge acts as the only master on the APB. All communication between masters on the AHB and slaves on the APB pass through this bridge. The APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. It is configured to connect five slaves: interrupt controller, timer, two UARTs, and parallel I/O port. The SpaceWire CODEC used with the SoC design is the core developed by University of Dundee for ESA [129].
4.2.2.1 Adding cores to the SoC

An important mechanism supporting the flexibility of the SoC design is the addition of IP cores to the design. A register file is used as an example to demonstrate the addition of IP cores to the SoC design. The register file is connected to the APB AMBA Bus. The AMBA APB Slave interface of the register file consumes a 1-kByte address space for its registers. The AHB/APB Bridge is responsible for the address mapping of the APB. The address-mapping configuration of the AMBA APB is described in a record in the VHDL file, device.vhd as shown in the following code listing.

```vhdl
constant apbslvcfg_tkconfig: apb_slv_config_vector(0 to APB_SLV_MAX-1) := (  
  --first last index enable function PADDR[10:0]  
  ("00000000000", "00000010000", 0, true), --memory ctrl, 0x00-0x08  
  ("00000010100", "00000011000", 2, true), --cache controller 0x14-0x18  
  ("01000000000", "01011111100", 13, false), --PCI arbiter 0x200-0x2FC  
  ("01100000000", "01111111100", 14, true), -- Register File 0x300-0x3FC  
  ("10000000000", "11111111100", 15, true), --DMA controller 0x400-0x7FC  
  others => apb_slv_config_void);  
```

Figure 4.4: Block Diagram of the System-On-Chip of the OBC Module
Chapter 4: Case-Study: Design of an On-Board Computer Module

The writing to and the reading from the register file is demonstrated using the following test program. The program was run on the SoC model using the simulation package called ModelSim. A snapshot of the results of the simulation is shown in figure 4.5.

```c
printf("test started");

*REGFILE0 = 0xDADCAD89; // write into reg0
*REGFILE1 = 0xBACDAC23; // write into reg1

value0 = *REGFILE0; // read from reg0
value1 = *REGFILE1; // read from reg1

printf("test ended");
```

![Figure 4.5: Simulation output of the writing and reading from register file added to the SoC.](image-url)

4.2.2.2 SpaceWire for the SoC Design

As stated in section 3.2, SpaceWire is chosen for the hardware modules of the module-based approach for SB-WSNs as the data interface between the modules in order to facilitate inter-operability between them. The purpose of the SpaceWire standard is to support the construction of high-performance onboard data handling systems, reducing system integration costs, promoting compatibility between data handling equipment and subsystems, and encouraging re-use of data handling equipment across several different missions. Use of the SpaceWire standard ensures that the equipment is compatible at both the component and sub-system levels. Processing units, mass-memory units and down-link telemetry systems using SpaceWire interfaces developed for one mission can be readily used on another mission reducing the cost of development, improving reliability and most importantly increasing the amount of scientific work that can be achieved within a limited budget.

SpaceWire is a lightweight, high-performance communications network designed specifically for use onboard spacecraft [130]. Its design offers an attractive option for
connecting together the different components of the on-board data-handling system of a spacecraft including processing units, mass memories, sensors, telemetry and telecommand subsystems efficiently. A Spacewire network is basically composed SpaceWire links, nodes and routers. Each node, which is the functional unit connected to the SpaceWire network, is fitted with one or more SpaceWire interfaces. The units are connected together either directly using point-to-point SpaceWire links or indirectly via spaceWire routers.

An important part component of a SpaceWire network is the SpaceWire CODEC which acts as an interface to the network. In order for the modules to be compatible with the SpaceWire network, and to be considered as a node in the network, the modules have to include a SpaceWire CODEC. The CODEC have three main components: the transmitter, the receiver and the state machine. The characters received by the CODEC from the node are converted into serial Data and Strobe output stream and sent down the network by the transmitter. The incoming serial stream is locked onto by the receiver, which then decodes the stream and outputs the decoded data in parallel form. The current operating mode of the CODEC is handled by the state machine. The operating mode is changed by errors, received characters, Time-outs and user requests. The state machine also provides link initialization, normal operation and error recovery services [89].

![Block diagram of the SpaceWire CODEC](image)

**Figure 4.6: Block diagram of the SpaceWire CODEC [89].**

### 4.3 Implementation Results

In this section, implementation results from the prototyping of the on-board computer are reported. There are three types of results that are required to evaluate the OBC module in the context of SB-WSN: Size, performance and interfacing. Size is important to ensure that
the SoC design does fit into the resources (logic and memory) that are made available from the design of the MCM design explained in the chapter. Performance results are required to ensure that the design is capable of satisfying the minimum processing requirements of the nodes of a SB-WSN. Interfacing is needed to allow the OBC module to be part of a distributed computing system. The first subsection reports the results of implementing the system-on-chip using FPGA technology. Results revealing the computing power of the OBC module are presented in the second subsection.

4.3.1 SoC Resource Utilization

This section gives details about the amount of resources utilized by the SoC design. Table 4.5 shows a summary of results showing estimated area requirements targeting the XC2V3000 FPGA after synthesis (using Synplify) and implementation (using Xilinx ISE). The table shows that the resource utilization of the SoC with the Leon-2 processor without any Input/Output peripherals occupies only 34% of the total number of slices that are available in the XC2V3000 FPGA. This consequently means that more components can be added within the SoC design, including additional LEON-2 processors to form a Multi-processor System-On-Chip. Adding more Input/Output cores to the SoC has only a significant effect on the Input/Output blocks utilization. This is because the capacity of the FPGA is huge and therefore the percentage of the occupied slices is insignificant.

<table>
<thead>
<tr>
<th>Table 4.5 Implementation results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XC2V3000 FPGA</strong></td>
</tr>
<tr>
<td>Slice Flip</td>
</tr>
<tr>
<td>LEON 2-1.0.30</td>
</tr>
<tr>
<td>LEON 2-1.0.30 + Ethernet</td>
</tr>
<tr>
<td>LEON 2-1.0.30 + Ethernet + PCI</td>
</tr>
</tbody>
</table>

The operation of the OBC Module involves the loading of the FPGA bit stream from the Configuration Proms of the MCM to the FPGA on power-up. Therefore it is important to make sure that the configuration files of the SoC design would fit into the available PROMs on-board. The size of the configuration files that are downloaded to the configuration PROMs are listed in table 4.6. The table shows that the configuration files consumes only a
small amount of PROM space available on the proposed OBC (0.5 Mbytes) and the emulation board (1.5 Mbytes). The maximum PROM consumption from the table is 2.63 Kbytes which is only 0.5% of the available PROM on the proposed OBC Module.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leon-2-1.20 + FPU</td>
<td>2.02 Kbytes</td>
</tr>
<tr>
<td>Leon-2-1.27 + Eth + Multiplier</td>
<td>2.44 Kbytes</td>
</tr>
<tr>
<td>Leon-2-1.24 + PCI + GRFPU + MMU + Multiplier</td>
<td>2.63 Kbytes</td>
</tr>
</tbody>
</table>

The memory consumption of some SoC software applications is summarized in table 4.7. There are two sets of programs given in the table: a set of programs that include the RTEMS operating system and a set of programs without RTEMS. Hello.c that is a simple program that prints out a “Hello World” message to the UART of the SoC consumes 45.3 Kbytes. The same program wrapped by the RTEMS operating system consumes 894.5 Kbytes of memory. DMA-echo is a program used to test the DMA controller of the SoC. It consumes 78.1 Kbytes. RTEMS-tasks is a program that tests the task switching ability of RTEMS, by alternating between 4 RTEMS tasks. It consumes 980.5 Kbytes. A quite representative networking application is the TTCP program [131]. TTCP (Test Transport Control Protocol) is a utility program for measuring network throughput. The TTCP program wrapped by the RTEMS operating system consumes an amount of memory of 2.0 Mbytes. With a total memory available for the module of 384 Mbyte (3 Gbits), TTCP + RTEMS consume around 0.52% of the total memory. This gives an indication to the suitability of the amount of memory available to the OBC module.
Table 4.7 Memory Consumption of programs for the SoC Design

<table>
<thead>
<tr>
<th>Program</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello.c</td>
<td>45.3 Kbytes</td>
</tr>
<tr>
<td>RTEMS-Hello.c</td>
<td>894.5 Kbytes</td>
</tr>
<tr>
<td>DMA-echo.c</td>
<td>78.1 Kbytes</td>
</tr>
<tr>
<td>RTEMS-TTCP.c</td>
<td>2.0 Mbytes</td>
</tr>
<tr>
<td>RTEMS-tasks.c</td>
<td>980.5 Kbytes</td>
</tr>
</tbody>
</table>

4.3.2 OBC Performance

The performance of the LEON processor was measured using the Dhrystone benchmark [132]. The results are shown in table 4.8. The number of “Dhrystones per second” is the time spent in the benchmark divided by the number of iterations of the benchmark. The performance is measured using three configurations. Firstly the benchmark was run on the LEON-2 processor clocked at 20 MHz without any operating system. The same benchmark was run on the LEON simulator TSIM. The Dhrystone benchmark was then run on the LEON processor clocked at 20 MHz with the RTEMS operating system.

Table 4.8 Performance of the LEON processor

<table>
<thead>
<tr>
<th>Platform</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEON-2 @ 20MHz</td>
<td>14742.0 Dhrystones per second</td>
</tr>
<tr>
<td>LEON-2 @ 20 MHz with RTEMS</td>
<td>7692.31 Dhrystones per second</td>
</tr>
</tbody>
</table>

The outcome of this comparison shows that the performance of the LEON processor with the RTEMS operating system running is around half the performance in the case of the processor without the RTEMS operating system. The reason for this is due to the operating system overhead, with only a single task being executed.
4.4 Conclusions

The main aim of this chapter was to demonstrate the modular concept that was proposed in section 3.2. In particular, it demonstrates the idea of hardware modules through a detailed study on the design of an on-board computer module.

The proposed design shows the possibility of designing modules that have the same specifications but with lower mass and power consumption compared to commercially available state-of-the-art on-board computers, using a combination of MCM and SoC technologies. Comparing the low-power OBC386 designed for microsatellite platforms with the designed on-board computer module, it is found that both incorporates a 32-bit processor and has memory of 256 MBytes. The proposed OBC Module would have a mass of 70 grams provides a significant amount of mass reduction compared with the OBC386 (1.7 Kg).

However, while the designed module meets the mass requirements of on-board computer modules for SB-WSN nodes it still does not meet the projected power requirements. Section 3.1 showed that the total power available for the nodes of SB-WSN is 2.65 Watts. Assuming that the OBC would have a power budget that is 10% of the total node power, gives an OBC maximum power rating of 0.265 Watts. The OBC module consumes 3.63 Watts of power. This is significantly short of the power requirement for the Module.

Previous work has been proposed that either use MCM technology (e.g. JPL's advanced flight computer) or SoC design (e.g. Astrium's Spacecraft Controller On Chip SCOC) for miniaturization but never combined both together. The approach used in the design of the OBC Module combining MCM technology and SoC in a single design allows the integration of the advantages of each of these technology in terms of both miniaturization and flexibility.

The OBC Module would be an integral part of a complete distributed system. There are two types of distributed systems that the OBC will be interacting with: the intra-satellite network and the inter-satellite network. The SpaceWire interface is the window via which the module interacts with the intra-satellite network that in turn, links it to the inter-satellite distributed system. The SpaceWire core was tested using ModelSim simulation tool.
Chapter 5

5 Data-Centricity for Space-Based Wireless Sensor Networks

While chapters 3 and 4 mostly covered the node-level aspects of SB-WSNs, this chapter is concerned with networking issues. Possible networking paradigms are evaluated for application in SB-WSN. A new data-centric networking protocol supporting distributed computing in SB-WSN is proposed and evaluated. Simulation software is developed for the purpose of evaluating distributed computing tasks using open-source tools. Section 5.1 provides an introduction to the concepts of data-centricity and data-centric networks. Section 5.2 discusses the details of simulation software that was developed for distributed computing evaluation. In section 5.3, a general evaluation of the data-centric design for networking in SB-WSN is given. Section 5.4 proposes and evaluates a novel data-centric mechanism tailored to SB-WSN. The conclusions of the chapter are given in section 5.5.

5.1 Introduction to Data-Centric Networks

5.1.1 Approaches to Satellite Networking

Several research works have been reported in the literature in relation to inter-satellite networking, which deal with different OSI layers, and in particular the lower layer protocols [133][134]. There is an obvious interest in the deployment of the TCP/IP protocol in space [135][136] for several reasons of which the most important is the compatibility with the Earth-based Internet and the level of maturity that this protocol has reached. However, the TCP/IP protocol, and the address-centric communication scheme in general, carry serious disadvantages to inter-node networking in SB-WSN. This is due to SB-WSN being formed from ultra-small satellite nodes being strictly limited in the resources that exist onboard. The overhead of using Internet networking is explained in reference [137]. An alternative to address-centric approaches is proposed. This approach, called the data-centric approach, is commonly used in WSNs research. This is the topic of the next section.
5.1.2 Data-Centricity in Wireless Sensor Networks

The traditional communication paradigm focuses on the relationship between the communicating peers i.e. the sender and the receiver of the data. In WSNs, on the other hand, the application is not interested in the identity of the nodes, but rather the actual interest lies in the information that the nodes possess about the physical environment that the WSN operates in. The concept behind data-centric networking is that the focal point of the network is the data being communicated and not the identity of the nodes. The consequence that this has to an application is that the requests to the network use data (and not nodes) as addresses [138].

A simple illustration of the data-centricity concept is shown in figure 5.1. In the address-centric approach, the data source (node 1) sends a data packet preceded by the address of the sink node (node 4). In the data-centric approach as shown in the right diagram of figure 5.1, the source sends the data packet preceded by an identification tag ("A" in this case). Only the nodes waiting for "A" will receive the data packet. The figure illustrates the one-to-many aspect of the data-centric concept. The one-to-many relationship potentially optimises bandwidth consumption and node resources, when several nodes simultaneously participate in a communication session which is common in SB-WSN. Data-centric addressing also enables one-to-one, many-to-one and many-to-many relationships within the network.

![Figure 5.1: Data-Centric and Address-Centric approaches to distributed system design](image)

The data-centric approach allows very different networking architectures compared to traditional, address-centric networks. Data-centricity in WSNs enables useful network properties such as the following [139]:

- Allows in-network aggregation which causes a reduction in the amount of traffic flowing in the network.
• Data-centric addressing enables simple expressions of communication relationships.

• Decoupling in time – data requests do not specify any timing details for the response, a property that is useful for event-detection sensing applications.

• Fault-tolerance – as the nodes are no more the focus of the network, the failure of a node has a limited effect on the network.

• Scalability - the addressing does not depend on the number of nodes in the network contrary to the other approaches such as TCP/IP where there is a specific number of bits per packet allocated to the destination address. In addition, by applying data aggregation the number of packets flowing through the network is significantly reduced. Also, the possibility of using localized algorithms such as clustering enhances the scalability of the system.

All of the advantages outlined above make the data-centric design attractive to WSNs where the aim is to achieve optimum usage of the limited resources available within a highly constrained environment. In addition, WSNs are naturally data-centric [18] [98] with the data being continuously collected and integrated from a large number of physically dispersed sensor nodes.

The differences between WSNs in general and SB-WSNs in particular were explained in section 3.1. A study on the effect that these differences have on the deployment of data-centric networking in SB-WSNs is presented section 5.2.

5.1.3 Data-Centricity for Wireless Sensor Networks

5.1.3.1 Implementations of Data-Centric Networks

There are several approaches to implementation of data-centric networks. Each approach implies a certain set of interfaces that would be useable by an application. This section describes two of the most important schemes: publish/subscribe and databases.

(1) Publish/Subscribe Scheme

The conceptual idea of the publish/subscribe paradigm is essentially very simple. All the nodes in the distributed system are connected to a “software bus”. Nodes make their data available publicly on the software bus via a “publish” action. Nodes that have previously announced on the software bus their interest in that kind of data that have been published using a “subscribe” action are then notified that the data is available on the bus [139]. This concept is illustrated in figure 5.2.
The publish/subscribe interaction pattern provides three essential properties concerning the relationship between providers and subscribers of information [138]:

- **Decoupling in space:** There is no need for Publishers and subscribers to be aware of each other.
- **Decoupling in time:** There is no dependency between the Publishing and notification of data, with the "software bus" provides intermediate storage.
- **Decoupling in flows:** Asynchronous interactions with the software bus can take place without any blocking.

![Publish-Subscribe system](image)

**Figure 5.2: Publish-Subscribe system**

There are several flavours of publish-subscribe systems based on the methodology used for addressing the data. These flavours are as follows [140]:

- **Group-based addressing** –

In the earliest Publish/Subscribe systems, each node, publisher or subscriber participates as a member in one or more predetermined groups. The nodes make subscriptions and publications only to the groups they are members of. This group-based approach leads to restricted access in the system. Subscribers cannot receive publications from some publishers since they have no group in common. This scheme does not support data-centric communication which is an essential feature of the Publish/Subscribe approach.

- **Subject-based addressing** – In subject-based (also called topic-based) Publish/Subscribe systems, the publications/subscriptions have a subject (also called a topic). The subject
Chapter 5: Data-Centricity for Space-Based Wireless Sensor Networks

belongs to a pre-defined namespace of subjects. Subscribers subscribe for subjects that they are interested in, and publishers publish messages with subjects. A typical example for topics is names of stocks traded at a stock exchange; when the price of a given stock changes, a notification for the corresponding topic is generated. The primary advantage of subject-based publish/subscribe systems is simplicity. Its main disadvantage is that it lacks flexibility.

- **Content-based addressing** – The later developments of Publish/Subscribe systems led to the content-based systems. Here the subscription matching criteria are extracted from the message content itself. The important advantage of the content-based approach is that it provides maximum flexibility in stating the subscription criteria. The subscriptions can be made more elaborate and composite now. The content-based approach is of considerable importance to the sensor network environment offering the required flexibility in specifying the subscription criteria. An example for such a predicate would be “Is fuel reported from satellites greater than 2 units?”. Complex predicates can be constructed by combining primitive predicates using standard logical operators (and, or, not) with the usual semantics [141].

(2) **Databases**

A different view of WSNs is to consider them as dynamic databases, which is completely different from the publish/subscribe view. The database conceptual view matches quite well with the data-centric approach in the design of WSNs. This is because quantifying a certain aspect of the physical environment that is surveyed by a WSN is equivalent to formulating queries for a database.

Two of the most representative sensor database systems are TinyDB [142] and Cougar [143]. In TinyDB, users specify a set of declarative queries that define the information to be gathered from the WSN. Queries indicate the type of readings to be obtained, including the subset of nodes the user is interested in, and any simple transformations to be performed over the collected data. They are specified using a language like a structured query language (SQL). A sample query could be expressed as follows:

```sql
SELECT AVG(temp, light)
FROM sensors
WHERE location in (0,0,100,100) AND light > 1000 lux
SAMPLE_PERIOD 10 seconds
```

TinyDB queries are specified on a PC and then the task of distributing it to the WSN is left to the query executor. The query is then disseminated and then the results are then returned.
to the query dissemination point in an energy-efficient manner using several techniques including in-network processing and cross-layer optimizations.

Queries in TinyDB are disseminated through the entire network and collected via a routing tree. The root node of the routing tree is the end point of the query, which is generally where the user that issued the query is located. Nodes within the routing tree maintain a parent–child relationship in order to properly propagate results to the root. Research into query processing techniques includes the design of an acquisitional query processor for data collection in WSNs. Information such as where, when, and how often data are physically collected and delivered, can be leveraged to significantly reduce the overall power consumption in the sensor network.

5.1.3.2 Data-Centric Routing

Multihop networks such as WSNs packets have to be relayed from the source node to the destination node through an intermediate node. It is the task of intermediate nodes to determine which neighboring nodes to forward an incoming packet that is not destined for it. This is usually done using routing tables that lists destination nodes against the most appropriate neighboring nodes. Routing tables are constructed and maintained by a set of rules that forms what is called as a routing protocol [138].

Several routing mechanisms have been proposed specifically for WSNs putting into consideration the unique characteristics of such networks [144]. Almost all of these routing protocols can be classified as data-centric, hierarchical or location-based although there are some distinct ones based on the network flow or quality of service (QoS) awareness. As this chapter is about data-centricity we will only be concerned with data-centric routing protocols. In data centric routing the routing decision is based on the name(s) associated with the data.

5.1.3.3 Data Aggregation

The real power of data-centricity lies in the ability to operate on the data while it is transported in the network. The simplest example of such in-network processing is aggregation. Data aggregation can be perceived as a set of automated methods of combining the data that comes from many sensor nodes into a set of meaningful information. In this context data aggregation can also be referred to as data fusion.

Using the following equation (whose derivation is shown in Appendix C) quantifies the advantages of data aggregation in the context of data-centric networks [145]:

$$\lim_{d \to \infty} FS = 1 - 1/k$$

(5-1)
The equation shows that if the distance between the sink and the sources is large compared to the distance between the sources, then the optimal data-centric protocol gives k-fold savings over the address-centric protocol. When there are 4 sources that are close together and far-away from the sink, then the address-centric protocol will have about 4 times as many transmissions, i.e. there are roughly 75% fewer transmissions with data aggregation. When there are 10 such sources, the gains are nearly 90% and so on.

The usefulness of this conclusion is important in two respects:

1. It proves the advantages of Data-centricity for SB-WSN over address-centricity for networks that does not have a single gateway to earth as is the case with the TechSat-21 mission [108]. This is because in SB-WSN, the distance between the nodes is much shorter than the distance between the nodes and the ground operator which maybe considered as the eventual SB-WSN sink.

2. It proves that data-centricity is particularly advantageous in missions involving a large number of nodes with few downlinks as is the case with the Magnetospheric Constellation mission [47].

5.2 Experimental Evaluation

This section reports the results from the simulations that was carried out in order to evaluate and compare the two main candidates for implementation of distributed computing in SB-WSNs i.e. data-centric and address-centric architectures. A publish/subscribe system is used to represent the data-centric architecture in the simulation whereas the address-centric architecture is represented by a client-server system. The reason behind this selection is that these two systems are the ones most commonly used in each of the two architecture classes under investigation. To carry out the simulation experiments a simulation tool was developed that models the different important aspects of a formation flying satellite mission in a distributed computing environment.

5.2.1 Simulator Design

Several discrete event simulators exist for simulating networked systems [Appendix B]. It appears that the main difference between the different available simulators is the trade-off that each makes between scalability and the abstraction level that the tool simulates. The simulators that were considered in the selection process are explained in Appendix-B.

OMNET++ [146] is selected as the simulation environment to be built-up on for the purpose of this research for two main reasons. Firstly, OMNET++ can be used to simulate
different kinds of modules relatively easily in C++. Secondly, it provides the appropriate level of abstraction for distributed computing simulations. The main features that OMNET++ possesses are:

- It is a discrete time simulator. The simulated objects communicate with each-other by exchanging messages at discrete moments of time;
- It is written in C++ and Tcl/Tk;
- Several graphical interfaces allow easy debugging and variables inspection. It also offers support for recording data vectors and scalars in output files;
- It offers support for parallel execution;
- Simulated objects are represented by modules. The modules can be simple or composed (depth of module nesting is not limited). The modules communicate by messages (sent directly or via gates). Each module description consists of an interface description and a behavior description;
- It includes several random generators (based on several probability distributions) with different starting seeds – a quite useful tool for simulating random satellite events that occur in the space environment;
- Simulations are easy to configure using .ini file. Batch execution of the same simulation for different parameters is also included.

However, OMNET++ lacks certain features that are needed for the successful simulation of space-based formation flying WSNs such as support for wireless communication and mobility. These two most important deficiencies and the workarounds that we attempted are described below.

**Wireless communication emulation** - OMNet++ does not contain a native support for wireless communication. The messages are sent on links connected to the ports of the various entities. The simulator was upgraded so that it is able to provide such support. The wireless connectivity is emulated by dynamically creating/destroying of the communication links between the entities based on: (1) their position in a two dimensional plane and (2) the concept of transmission range. This emulated model is easy to implement and runs very fast. A full channel model can also be easily implemented, but unfortunately it will slow down the simulation process. At the simulator level, a central component holds the connectivity matrix of the entire system.

**Mobility framework** - Simulation models for formation-flying satellite missions are non-existent in the OMNET++. Simulating mobility in OMNET++ is not a straightforward
problem, as the existing mobility framework does only models basic mobility patterns such as circular movements. For this purpose a new mobility generation tool was developed. The tool is based on the generation of a mobility file that the simulator reads at run-time. The mobility file contains the data about the positions of the respective nodes at equally spaced periods of time. The mobility pattern is included in the design of the generation tool based on the orbital configuration required. The mobility file is generated before the actual simulation and can be reused, saving important simulation time. The mobility generation tool can be enhanced to include more features enabling a more sophisticated simulation of relative motion between the satellites in orbit with a larger library of mobility files representing various mobility configurations. While the mobility generation tool does not model the exact absolute positions of the satellites in orbit, it does fulfill our purpose of modeling the pattern of connectivity between the satellites in space – which is the main important factor from a distributed computing perspective.

The block diagram of the developed simulator is shown in figure 5.3. The developed simulator uses a central manager that takes care of storing the connectivity map and updating it on request. It also takes care of reading additional configuration files and sending the information to each satellite node. Each satellite node in the simulator is a compound module formed from a number of simple modules as shown in figure 5.3. The application module includes the implementation of the protocol that is being simulated. The layer0 module is the interface between each sensor node and the central manager. When the application layer wants to send a message to the neighbours within the transmission range it just forwards this message to the layer0 module. The sensor module (representing the payload of the node and the energy modules (representing the net amount of power available for the node) are currently not implemented and are put in the architecture for future expansion of the simulator.
5.2.2 Experimental Setup

Both of the publish/subscribe and client-server paradigms are implemented in the application layer (application.cc file) as stated in the previous section. Each of the two implementations are outlined in this subsection.

5.2.2.1 The Publish/Subscribe Implementation: Directed Diffusion

The Publish/Subscribe system that is chosen for the purpose of the evaluation is Directed Diffusion. Directed Diffusion is one of the pioneering data-centric communication paradigms developed specifically for WSNs [147]. Diffusion is based on a publish/subscribe API where the details of how published data is delivered to subscribers are hidden from the data producers (sources) and publishers (sinks).

Several protocol variants of directed diffusion exists each optimised for a different situation. This makes directed diffusion more of a design philosophy than it is concrete protocol [148]. The basic variant of directed diffusion is the “two-phase pull diffusion”
which consists of three phases as shown in figure 5.4: interest propagation, data propagation, and reinforcement. The first phase involves nodes broadcasting their interests in certain kinds of named data expressed in the form of a set of attribute-value pairs as shown in the following example.

```c
// detect reconfiguration of the satellite formation
type = position
// send back results every 20ms
interval = 20s
// for the next 5 minutes
duration = 5minutes
// from satellites with power available greater than 1.5Watts
power => 1.5W
```

Interest messages are distributed through the network either using flooding or some other more sophisticated technique. When a node receives the interest packet, it checks if the packet is new to the node by retrieving the internal cache of the node. If the packet is new to the node, the packet is cached and rebroadcast to the neighbouring nodes. The node also remembers which neighbouring nodes it received the interest packet from such that, later on once the data has been published, the actual data could be forwarded to all these nodes. This is called the setting up of a gradient toward the sender of an interest. A gradient cache is maintained at each node in order to store a separate set of gradients for each type of data received in an interest.

The second phase of the directed diffusion process involves the propagation of the data packets through the network and is initiated once the gradients are set up. A node that possesses the actual data required by the sink becomes a source and starts to send data packets. Each node that receives a data packet performs a matching operation according to a list of attributes and their corresponding values. If a match is established the node packet is passed on to the application module of the node, else the node is considered as an intermediate node.

In its simplest form, intermediate nodes would broadcast all incoming data packets to all their outgoing gradients, while possibly suppressing some of the data messages to adapt to the data rate of each gradient. A problem with this simple scheme is that it results in unnecessary overhead in the network as the data packets are needlessly repeated due to the presence of loops in the gradient graph. Simply checking the source of these data messages is not feasible due to the lack of globally unique identifiers. The problem could be solved by introducing a data cache at each node in the network. The data cache at each node stores the recently received data messages for each known interest. If the sink has multiple neighbours it reinforces one of its neighbours (for example the one which delivered the first
copy of the data message). To do this, the sink reinforces the preferred neighbour, which in turn, reinforces its preferred upstream neighbour, and so on.

![Figure 5.4: Three steps in directed diffusion (a) Interest propagation; (b) Gradient setup; (c) Data delivery along reinforced path.]

The fact that the variant of directed diffusion explained above involves two phases (first flooding the interest message through the network and then having data flow from the sink toward the source along a reinforce path), in addition to the fact that it is the sink that initiates the “pulling” of data, is the reason behind calling it “two-phase pull” directed diffusion. Other variants of the original form of directed diffusion explained above have been developed [147]. One such alternative is push diffusion, which is intended for networks with many receivers and only a few senders. A typical example is an application where sensor nodes need to subscribe to each other frequently in order to be aware of local events, but where the amount of actual events is quite low.

One-Phase Pull diffusion is another variant of directed diffusion [149]. This one is geared towards networks with many senders and a small number of receivers. As the name indicates, one-phase pull eliminates one of the flooding phases of two-phase pull, which constitute its major overhead. The network is still flooded with interest messages during the first phase of the procedure, however the interest messages set up direct parent-child relationships in the network between a node and the node from which it first receives an interest message forming a tree in the network.

5.2.2.2 The Client-server Implementation

The basic client-server interaction model that is implemented is shown in figure 5.5. Several derivations of this basic model exist (for example the client-server model based on sockets as shown in figure 5.5 (right)). The client requesting a connection to be made to the
server initiates the communications session. A session is started once the server acknowledges the connection request message. The client then sends a data query message to the server which responds by sending the required data. Finally the session is terminating by the client sending a disconnection request message and the server acknowledging the disconnection. A problem that might rise with this type of communication is that it is synchronous and therefore the failure of one of the communicating nodes can lead to deadlock. This could be overcome by using timers to terminate the session. The deadlock situation is out of the scope of this comparison between client-server and publish-subscribe.

![Diagram of client-server interaction model](image)

**Figure 5.5: The basic client server interaction model (left) and its socket-based derivative (right)**

### 5.2.3 Impact of Mobility

A unique aspect of picosatellite networks that differentiates them from other WSNs is the mobility of the nodes. Research in WSNs has mostly focused on static nodes and issues related to mobility are not well addressed. Picosatellite networks are different from ‘mainstream’ WSNs, in that the nodes here are mobile and their mobility is highly predictable.

There exist several formation flying configurations for picosatellite networks, each with its own set of advantages and disadvantages [72]. The simplest configuration is the one called leader-follower configuration (also called string-of-pearls) where the satellites have the same orbital parameters, but are available for communication at different times. The same-ground-track formation, termed “ideal” by NASA Goddard [5], is the one in which two or
more satellites have identical ground tracks. Simulations of the ranges of both types of formation-flying configurations are shown in figures 3.2 and 3.3, section 3.1.2.

Of importance to the OMNET++ simulator is the pattern of relative distances between the satellites so that the pattern of the connectivity between the satellites can be modelled. This pattern was extracted from simulations carried out using the STK simulation tool. Each of the configurations is obtained by adjusting the orbital parameters appropriately to obtain the required configuration.

As may be observed from figure 3.2 in section 3.1.2, the space-based formation flying missions with a leader-follower configuration are similar in behaviour (from mobility prospective) to the static WSNs that are common in terrestrial applications. Of interest to us is the same ground track configuration as an illustrative example of the behaviour of the distributed computing paradigms in the context of the relative mobility of the satellites in the formation. The relative motion of the same ground track configuration was implemented in the simulator as an additional feature in the OMNET++ simulator.

An experiment was carried out in which the success ratio of the data delivery (Number of successfully delivered data messages/Total number of data messages sent) was compared between the basic publish/subscribe implementation in OMNET++ and that of the client-server. In the publish/subscribe model, a subscription is made at simulation time of 6 seconds to cause a publication from the source to take place for an interval of 10 simulation seconds at a frequency of one publication every simulation second. In the client-server model, on the other hand, a data request is made for 10 seconds starting from the 6th simulation second. The experiment was repeated several times, each time increasing the number of satellites in the formation. The results (shown in figure 5.6) indicate that while the client-server model does not suffer from any packet losses, the basic publish/subscribe model does. This is because, in the publish/subscribe model, the gradient between the source and the sink is set based on the satellite positions at the time when the subscription message is sent from the sink. It does not take into account the dynamics of the formation. It follows from that, that the original implementation of directed diffusion is not suitable for SB-WSN. The protocol needs to be adapted appropriately to overcome this deficiency.
An experiment was carried out to determine the variation of the number of hops a data message requires to reach the sink with the time of the data subscription initiated by the sink. The experiment was carried out with 15 nodes in the network. A subscription message is sent at 6 simulation seconds and the number of hops the interest message takes to reach the source is noted. The experiment is repeated changing the subscription time. The simulation results presented in figure 5.7 shows that the number of hops, varies with the time in which the subscription message is sent. Since the number of hops is proportional to the communications energy consumption, it can be concluded from figure 5.7 that communications power and bandwidth may be saved by appropriately adjusting the publication and subscription times. This could be used to design an energy-efficient protocol for data dissemination in picosatellite networks.
Figure 5.7: The variation of the number of hops between the source and sink with the time of sending the subscription message

5.2.4 Scalability

Scalability is an important evaluation factor for SB-WSNs. This is due to the fact that future space missions are going to include tens, hundreds or even thousands of Pico-satellites flying in close formations [77]. The experimental scenario for the scalability evaluation was as follows. A subscription message (data request in the client-server case) is sent after 6 seconds of simulation time from node 0. The reply from the data source (publications in case of the publish/subscribe paradigm and data result messages in case of the client-server paradigm) was sent every 1 simulation second until the end of the simulation at 18 simulation seconds. It should be noted that the client-server results was based on having a connection session each time a data packet is sent. Figure 5.8 show that client/server does not lack scalability with a direct proportionality relationship between the number of packets sent and the number of nodes in the network. However the publish-subscribe system has a much lower gradient than the client/server system. This leads to a conclusion that the scalability of publish/subscribe is much more than client/server.

5.2.5 Discussion

The simulation results presented in section 5.3 show that the publish/subscribe mechanism possess significant advantages such as being more scalable than the client-server mechanism. It is observed from figure 5.10 that the publish/subscribe paradigm in its simplest form is scalable contrary to the client-server approach that requires several control messages to be exchanged besides the actual data.

However, the original publish/subscribe mechanism needs to be modified in order to be able to work in the conditions imposed by the space environment. In particular, the problem
is caused due to the constant relative mobility between the nodes as shown in the results in figure 5.8. This is because the mobility of the nodes causes a high rate of packet loss under the publish/subscribe approach which does not adopt any acknowledgement scheme, contrary to the client-server approach.

5.3 Data-Centric Protocol Design

Following from the evaluation of the data-centric approach to networking in SB-WSN in section 5.2, a novel protocol is proposed, which is tailored for SB-WSN. This mechanism overcomes the impact of mobility on the performance of SB-WSNs.

5.3.1 Assumptions

The following assumptions are made for the design of the data-centric system:

- The nodes in the network exchange formation-keeping messages constantly at small periods of time in order to avoid collisions. Figure 5.9 shows the minimum sampling time required for updating the knowledge of the node about the positioning of the neighbouring nodes in order to avoid collisions. This variable is important in the design of the publish/subscribe mechanism as will be seen in the next section.
- All links are bidirectional and no control messages are lost.
A broadcasting network is assumed (due to having omnidirectional antenna onboard the satellite nodes for inter-node communications).

It is also assumed that each of the nodes in the network has its own unique identifier (for example, a MAC address).

It is also assumed that each satellite knows its relative position within the cluster, possibly by carrying a suitable GPS receiver on board.

![Figure 5.9: Variation of the sampling time requirement with the accuracy of the GPS receiver (shown in the figure as "e") and the distance between the nodes (figure reproduced from [150])]()

### 5.3.2 Protocol Design

The main goal of the designed protocol is to find the most appropriate way of passing data from a source satellite to a sink satellite overcoming the mobility problem that was identified section 5.2. The sink (also called subscriber) broadcasts an interest message, specifying their interest in a certain kind of data by a set of attribute-value pairs. The novelty of our protocol comes from the introduction of the concept of the time gradient in order to solve the problem identified in section 5.2.1 that is relevant to the impact of mobility on the satellite nodes. We accomplish this with a 3-phase protocol:

- The first phase establishes basic information at each node using the formation keeping messages that are sent within the network in order to avoid collisions between the nodes.
Chapter 5: Data-Centricity for Space-Based Wireless Sensor Networks

- The second phase involves the sink sending out an interest message. A gradient is set up during this phase between the sink and the source using data stored in a Neighbour Information Table (NIT).

- The third phase is the one in which the data packets are routed along the best available route.

5.3.2.1 Phase I: Initialization

The first phase of the protocol takes one orbit after a reconfiguration of the formation takes place. Each node sends a 1-hop “Formation-Keeping” (FK) Message every 5 minutes. This message is received at the neighbour of the node. Each node updates its neighbour information table based on the data it receives through the FK messages. The structure of the Neighbour Information Table consists of two fields: the Neighbour_ID and Contact_Time. A more sophisticated NIT is possible in order to allow further optimization of the routing process by including more data about the Neighbours. This however would require modification of the FK message structure to include the data that would be needed to update those fields within the NIT.

5.3.2.2 Phase II: Interest Announcement and Gradient Setup

The second phase of the protocol is initiated from a sink node that desires to receive data of a particular type to broadcast its request in the form of an interest packet to the entire network.

1) The interest message:

The interest message specifies the interest of a subscribing node in a certain kind of data using a set of attribute-value pairs. The interest message structure is shown in figure 5.10. The interest packet is divided into four sections. The first section includes fixed fields that remain unchanged during the lifetime of the interest packet. The combination of the sinkID and the ISeqNum forms unique ID for the packet. The frequency and interval fields represent the rate of publication and the duration for which the data is required respectively. The second section includes the fields that are updated by each node in the network. The TTL (Time To Live) field is decremented by one at each node. In particular, it is used for real-time traffic to limit the overall path length. As larger TTL would mean larger end-to-end delay between the source and the sink, TTL needs to be appropriately set by the sink node. The TimeStamp field is of particular importance for satisfying the real-time requirements of communications.
The third section contains information about the data itself that is required. The naming scheme that is used in the design is called the attribute-naming scheme [151]. The attribute-naming scheme is important for the matching process that takes place at the intermediate nodes of the network. The naming scheme is based on attributes that consist of three fields: key, operator and value. The key is equivalent to the Topic in topic-based publish-subscribe systems (introduced in section 5.1.3.1). It is used to identify the nature of the required data. The operation field identifies how data messages and interests interact. Operations are usually binary comparisons (e.g. EQ, GT, LT corresponding to equal to, greater than and less than respectively). The value field indicates the value that the operator needs to satisfy during the matching process.

The fourth section of the interest message includes details of the QoS that needs to be achieved. While an interest message may include a number of QoS criteria, two of the important ones are shown in figure 5.10: QoS_deadline and QoS_energy.

![Figure 5.10: The Interest Packet](image)

2) Gradient Setup

The gradient set up process is illustrated in figure 5.11. The original directed diffusion mechanism [147] (explained in section 5.3.2.1) uses the path with the least latency to setup gradients. We introduce a new type of gradient: the time gradient. The purpose of the time gradient is to choose the most suitable time for forwarding a time message taking into account Quality-of-Service (QoS) requirements identified by the application. Therefore the
gradient is formed from a combination of the ID of the node to send the data message next and the time the message may be sent.

5.3.2.3 Phase III: Data Gathering

When an interest message is received at a node which contains the data requested in the interest message, a data message is released and is passed along the gradient that has been previously set. The data message is straightforward and is shown in figure 5.12. The most important information it contains is the identifier of the sink node and the TTL for the packet, indicating the life-time of the data packet in terms of the number of hops. Each node that receives the packet decrements its TTL field. The packet is discarded when its TTL field is 0.
5.4 Data-Centric Protocol Experimentation

In this section we report the results of experimental work carried out on the data-centric scheme that was proposed in section 5.4.

5.4.1 Functionality Verification

An experiment was designed using the simulator that verifies the basic functionality of the data-centric protocol that was explained in the previous section. The experimental scenario involves a SB-WSN formation of 15 nodes flying in a same-ground track formation. The simulation time is 28 simulation seconds.

This experiment was repeated for several subscription times. The results are shown in table 5.1. The data message was received successfully at most of the subscription times. In certain cases, for example when the subscription message was sent at a simulation time of 10 simulation seconds, the data message is not received by the sink. The reason behind this is that at that simulation time, the sink (in this case node 4) is out of range of any other node at the time of when the subscription message was sent.

The snapshot in figure 5.13 is taken from the output screen of the simulator for the case when the subscription message is sent at time of 8 simulation seconds. The figure shows that the data message is successfully received at the sink node (which is node 4 in this case). The data message consists of three attribute-value pairs. The figure also shows the contents of the NIT of each node at the end of the simulations.
Table 5.1: Results from the functionality verification experiment

<table>
<thead>
<tr>
<th>Subscription time (Simulation seconds)</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>No</td>
</tr>
<tr>
<td>14</td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>Yes</td>
</tr>
<tr>
<td>18</td>
<td>Yes</td>
</tr>
<tr>
<td>20</td>
<td>Yes</td>
</tr>
<tr>
<td>22</td>
<td>Yes</td>
</tr>
<tr>
<td>24</td>
<td>No</td>
</tr>
<tr>
<td>26</td>
<td>Yes</td>
</tr>
<tr>
<td>28</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 5.13: Output of the simulator verifying the operation of the mechanism

5.4.2 Performance Measurements

The following parameters are used for evaluating the performance of the proposed protocol:
Percentage overhead of number of packets: The ratio of the number of messages sent during the experiment to the number of messages sent without any publications being made in the network.

Averaged delay: The delay from sending an interest packet to receiving the data.

The simulation settings are as shown in table 5.2 except if mentioned otherwise. These settings are chosen as they represent the situation of a typical SB-WSN mission.

Table 5.2: Simulation Settings for the Evaluation Experiments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes</td>
<td>34</td>
</tr>
<tr>
<td>Simulation time</td>
<td>32 simulation seconds</td>
</tr>
<tr>
<td>Interval between Publications</td>
<td>2 simulation seconds</td>
</tr>
<tr>
<td>Time To Live (TTL)</td>
<td>8 Hops</td>
</tr>
</tbody>
</table>

5.4.3 Effect of Distances between the Nodes

In this experiment the overhead percentage was monitored while varying the distances between the nodes. The term overhead here is used to describe the increase in the number of packets flowing in the network in the case of having one or more publication compared to the case when there are no publications. Figure 5.14 shows that the increase in the distances between the nodes leads to an increase in the number of messages being sent in the network. This increase gradually decreases until it saturates at a particular point. The interpretation of this behaviour is that with the nodes being closer the lengths of the gradients are shorter and therefore the number of messages to be sent per gradient is less. The delay was also monitored, but due to the nature of the protocol the delay was found to be constant despite the variation of the distances between the nodes in the network.
5.5 Conclusions

An important design decision that needs to be made for SB-WSNs is the interaction pattern between the nodes. Using data as the central object of the network rather than the traditional approach of using the nodes themselves incurs several advantages for SB-WSNs of which the most important is the ability to do in-network processing that would lead to a reduction in the number of packets flowing within the network. SB-WSNs have certain characteristics that differentiate them from the mainstream WSNs. These characteristics need to be taken into account when incorporating data-centricity to the network. These characteristics include mobility, complexity of the nodes and the control system factor.

The simulations that were presented in this chapter show that the state-of-the-art protocols cannot be applied directly to the SB-WSNs without alterations. This is due to several factors of which the most important is the high packet loss rate that the network experiences due to the relative mobility between the nodes as was shown by the simulations. For this reason a novel data-centric protocol was designed to overcome this problem. The proposed data-centric protocol is the first data-centric protocol to be designed specifically for SB-WSN. The protocol overcomes the mobility problem by
utilizing the unique feature of SB-WSN of having the ability of constructing NITs to form time gradients.
Chapter 6

6 Design of Middleware for Inter-Spacecraft Networking

Although data-centric protocols are effective in the optimal usage of resources in SB-WSNs, the gap between the protocols and the application inhibits their effective use by application developers [152]. To make these protocols more useful, application developers would benefit from a middleware layer that hides the details of these protocols, while providing an API (Application Programming Interface) that reduces the cost of developing applications. This chapter demonstrates the practical implementation of a data-centric architecture using the design of a middleware layer that is tailored for SB-WSNs. The developed middleware software is referred to as Middleware for Inter-Spacecraft Applications (MISA). The middleware also demonstrates the design of software modules for SB-WSNs.

This chapter presents the details of the design of the MISA middleware. Section 6.1 presents background information and survey on middleware. Details about the architectural design of MISA are given in section 6.2. This is followed by a description of the implementation of the middleware in section 6.3. Section 6.4 illustrates the usage of the middleware for SB-WSN.

6.1 Review on Middleware

6.1.1 Introduction

Software modules in distributed systems are distributed across separate processors, causing the possible interaction mechanisms that take place between the modules to be significantly increased when compared to a single processor system. This leads to a significant increase in the complexity in the design and implementation of the software, as each software interconnection must be handled individually. Furthermore, extensive modification to the existing would be required to incorporate additional functionality to the software. This problem is the problem that middleware addresses.
Middleware refers to software that lies logically between the operating system and the user’s software applications. Its main responsibility lies in managing interconnections between the user’s software applications. A standardized interaction mechanism abstracts away some details of the connection. Middleware therefore provides interface transparency to the user. Thus, application designers need to only specify that Module-A is connected to Module-B without needing to worry about the precise implementation, or which processors the modules run on.

The middleware layer is required to provide [153]:

- Standardized system services to diverse applications
- A run-time environment that can support and coordinate multiple applications
- Mechanisms to achieve adaptive and efficient resource utilization of system resources

The aim of middleware for SB-WSNs is to provide appropriate abstractions and mechanisms for the application developer. When developing distributed applications, designers do not have to explicitly deal with problems related to distribution, such as heterogeneity, scalability, resource sharing and fault tolerance. Middleware developed upon network operating systems provides application designers with a higher level of abstraction, hiding the complexity introduced by distribution. This reduces the software costs and increases reliability.

6.1.2 Middleware Classification

With the large number of middleware that exist, it seems a difficult task to make a classification of middleware technology. Schmidt et al. [154] proposed a classification that involves a vertical classification of middleware based on its functionality, similar to the OSI networking layered classification as shown in figure 6.1. Any middleware might lie in one of these layers or it might extend its functionality to several layers. The middleware layers are as follows:

- **Host infrastructure layer**: This layer eliminates tedious and error-prone aspects of developing and maintaining distributed applications that use low-level network programming mechanisms (such as sockets) by incorporating core operating system communication and concurrency services. Examples of infrastructure middleware are Java Virtual Machine (JVM) [155] and the Adaptive Communication Environment (ACE) middleware [156].
• **Distribution layer**: This layer builds upon the lower-level infrastructure middleware. Its purpose is to automate common network programming tasks, such as parameter marshaling/demarshaling, socket and request demultiplexing, and fault detection/recovery. Prominent examples of distribution middleware are CORBA [157] and DCOM [158].

• **Common Services layer**: This layer defines domain-independent services, such as multimedia streaming, logging, multimedia streaming, event notification, security, transactions, fault tolerance, and distributed concurrency control. Example of common services middleware is the CORBA Event Service [159].

• **Domain Services layer**: This layer is tailored to the requirements of particular domains, such as telecommunications, e-commerce, health-care, or process automation. Example of middleware in this layer is ESA’s Packet Utilization Standard (PUS) [160].

![Figure 6.1: Middleware Layers [156]](image)

The distribution middleware is commonly referred as today's middleware. Most important developments in the field are in this layer. Schantz et al. [154] consider this middleware layer as the one defining higher-level distributed programming models whose reusable APIs and components automate and extend the native OS network programming capabilities encapsulated by host infrastructure middleware. A classification of distribution middleware is shown in figure 6.2.
Chapter 6: Design of Middleware for Inter-Spacecraft Networking

Middleware

- Message-oriented
- Remote Procedure Call (RPC)
  - Distributed shared memory
  - Sun RPC [162]
  - LINDA [163]

- Address-Centric model
- Data-centric model
- Remote method invocation (RMI)
  - Network services
  - Object Request broker
  - Java RMI [164]
  - JINI [165]
  - CORBA [166]

- Object-Agent [5]
- Data Distribution Service (DDS) [161]

Figure 6.2: Classification of Middleware Software

The Remote Procedure Call (RPC) model [167] extends the simple local procedure call interface to offer the ability of invoking a procedure whose body is across a network. The RPC interaction takes place between two independent processes that may reside on the same or different processors. The operation of RPC is as follows. Process A, makes a request to another process, B, by issuing a procedure call to B and passing with the call a list of argument values. B then executes the procedure as in the case of local procedure calls. As observed from the above explanation, the RPC model is action-oriented with the data passed as arguments. This is different from the message-passing model that is data-oriented, with the actions triggered by exchanged messages.

Distributed Object middleware [158] provides the abstraction of an object that resides on a remote processor while its methods can be invoked just like those of an object in the same address space as the caller. All the software engineering benefits of object-oriented techniques—encapsulation, inheritance, and polymorphism—available to the developer of distributed applications by using distributed objects. Remote Method Invocation (RMI) is an object-oriented extension of RPC. In this model, a process invokes the methods in an object which may reside in a remote host.

The network services paradigm is essentially an extension of RMI. The difference between the two is that for the network services paradigm the service objects are registered with a global directory service. This allows the objects to be looked up and accessed by service requestors on a federated network. In the ORB paradigm, an application issues requested to an object request broker, which directs the request to an appropriate object that provides the desired service. The ORB paradigm is the basis of one of the most common middleware
Chapter 6: Design of Middleware for Inter-Spacecraft Networking

systems, CORBA [166]. CORBA hides the location of remote objects, simplifying the application’s interactions with these remote objects by allowing all operations to appear local.

Message-oriented middleware (MOM) increases the flexibility by enabling applications to exchange messages with other programs without the need of knowing what platform or processor the other application resides on within the network. At the core of the MOM architecture is a message system that acts as a switch for messages, through which processes exchange messages asynchronously in a decoupled manner.

The client-server model for message passing is the most popular architecture in distributed systems and consequently in middleware design. Client-server networks include servers - machines that store data, and clients - machines that request data. An alternative that have recently emerged as a powerful alternative to the client-server model in some applications (particularly embedded systems) is the data-centric model represented by the publish/subscribe paradigm discussed in section 5.1. A good example of a middleware that adopts this model is the Data Distribution Service (DDS) [161]. Data-centricity refers to the fact that it is the data that is the centre of operation rather than the nodes as is the case with the address-centric approach that is associated with the client-server model. A comparative analysis of these two models was presented in chapter 5.

6.1.3 Middleware for Wireless Sensor Networks

Recently, a number of projects have targeted the development of middleware specifically designed to meet the challenges of WSNs, focusing on the long-lived and resource-constrained aspects of these systems. Romer et al. [168] clearly states the purpose of middleware for WSNs to include all aspects that supports the building of the application software of WSNs including development, maintenance, deployment and execution. This general definition of the functions of WSN middleware encompasses many tasks such as the following [168]:

- Providing appropriate APIs for expressing complex high-level tasks,
- Communicating the tasks to the WSN,
- Coordinating the splitting of the task and distributing the outcome of this task division to the individual nodes,
- Performing data fusion of the readings of the individual nodes in order to produce a high-level result, and
- Reporting the results back to the task issuer.
A classification of WSN middleware according to the core idea that the design of the middleware is centered on is shown in table 6.1.

**Table 6.1: Middleware Classification**

<table>
<thead>
<tr>
<th>Category</th>
<th>Explanation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual machine</td>
<td>Virtual machines hide the heterogeneity of distributed systems by providing a low-level abstraction to the hardware and the run-time environment of the nodes. This allows the use of an efficient programming paradigm in the development of distributed algorithms.</td>
<td>Mate [169]</td>
</tr>
<tr>
<td>Database-based</td>
<td>The database approach treats the whole sensor network as a large &quot;virtual database. Interaction with the sensors is done in form of system queries using SQL like language.</td>
<td>SINA [170]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cougar [143][171]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFuse [172]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TinyDB [142]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TinyLime [173]</td>
</tr>
<tr>
<td>Event-based</td>
<td>The event-based middleware uses an event triggered approach where the execution is driven by the events. Event-driven communication is an asynchronous paradigm that decouples senders and receivers event triggered. Middleware adopting this type of architecture are mostly based on the publish/subscribe paradigm.</td>
<td>Mires [174]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSWare [175]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Impala [176]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Envirotack [177]</td>
</tr>
<tr>
<td>QoS-based</td>
<td>This approach is used provide data based on the Quality of Service (QoS) required that is provided by the application developer to the middleware.</td>
<td>MILAN [178]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AutoSec [179]</td>
</tr>
<tr>
<td>Agent-based</td>
<td>This category of middleware provides the application developer with an agent development framework. Applications are designed in the form of agents.</td>
<td>RUNES [180]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SensorWare [181]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Agilla [182]</td>
</tr>
</tbody>
</table>

Although table 6.1 shows categorization of middleware architectures based on the main concept that the design revolves around, these concepts can exist in any type of the middleware architectures. For example an event-based middleware may provide the application developer with the option of including QoS in their software design.

Due to the advantages of publish/subscribe systems that were explored and identified in chapter 5, the architectural design of the MISA middleware lies in the event-based architecture category.
6.1.4 Middleware for Space Applications

Some work has been carried out on the design of middleware for space applications as will be seen in this section. However most of this research is unsystematic and was either done by computer scientists or space engineers who over emphasize a particular aspect of the research topic and therefore their research ignores certain aspects of distributed computing for space. A few projects that claim to involve distributed computing for space are outlined in this section.

ObjectAgent

Princeton Satellite Systems have developed the Object-Agent (OA) middleware under contract from the U.S. Air Force Research Laboratory (AFRL) for the cancelled TechSat-21 mission [5]. Object-Agent is an agent-based software architecture framework where agents are used to implement all of the software functionality and communicate through simplified natural language messages. While OA has been used to demonstrate cluster formation flying and collision avoidance algorithms, its role is to simply provide a framework for designing mission software using agents. It does not provide a comprehensive solution to space middleware. It does not take into consideration any resource constraints and therefore no results were published evaluating its performance. Although OA acknowledges the need for real-time, it assumes that all real-time activities can be encapsulated within the middleware modules (i.e. within the agents).

Advanced Avionics Architecture and Modules

The Advanced Avionic Architecture and Modules (A3M) funded by the European Space Agency (ESA) is aimed at developing a middleware layer to enforce fault-tolerance to high-performance computing on-board satellite missions [183]. It particularly touches upon the development of two algorithms namely: Uniform Consensus (UCS) and Uniform Coordination (UCN) to solve problems such as replicated processing and distributed access to shared resources that are common in distributed computing systems.

The application of the A3M project is limited to areas that require the UCS algorithm such as parallel cluster computing on a single spacecraft or for the development of Fault Detection, Isolation and Recovery (FDIR) mechanisms for formation flying. As the A3M focus is different than that of this research work its outcome is also very different from the outcome of this research.

Spacecraft On-board Interfaces and Service

The Spacecraft On-board Interfaces and services (SOIS) is a standardization effort by the Consultative Committee for Space Data Systems (CCSDS). SOIS defines generic interfaces
and services that would simplify the way that flight software can access flight hardware. This would lead to the improvement of the spacecraft flight segment data system design and its development process. The SOIS architecture, defines a standard set of services for use onboard spacecraft [184] [185]. A stack of layers categorizes the services where each of the layers provides a specific set of functionality.

Some work has been done in building middleware for spacecraft using SOIS. The Service Layer SoftWare (SLSW) is a middleware architecture that takes the SOIS recommendations as an input to provide standardized services to the application software, disregarding the details of the underlying hardware and software services. The SLSW architecture has its services organized into three main groups: communication services, Command and data acquisition services and other services.

The SOIS standard aims at simplifying software design of a single spacecraft platform by standardizing the interfaces between the various computing entities and communications buses. It does not consider the software architecture for formation flying missions. In addition, the SOIS is based on an address-centric architecture. In this thesis a data-centric architecture for both inter-spacecraft and intra-spacecraft applications is proposed. While the focus of this work is on inter-spacecraft applications, the data-centric architecture could be extended to include intra-spacecraft applications in the future.

### 6.2 Middleware Design

#### 6.2.1 Design Requirements

The design requirements of middleware for SB-WSNs have been identified as follows:

1) *Modularity/Component-based*

The middleware forms part of the module-based view (see section 3.2) of SB-WSNs. A component-based model for the middleware is therefore essential. The motivation behind a component-based model is that such a model enhances the modularity and facilitates an easy composition of the middleware. This will lead to reduction in the mission software costs in parallel to reduction in the hardware costs. A component-based model is essential for embedded systems as it increases reliability without sacrificing performance. The component model allows an application developer to be able to easily combine independent components into an application specific configuration.
(2) **Real-time Support**

A system is said to be *real-time* if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed \[186\]. The middleware design should support deterministic real-time performance.

The approach used in the middleware design is to provide support for Quality of Service specification.

(3) **Built-in FDIR mechanisms**

Fault-tolerance is the ability of a system to deliver a desired level of functionality in the presence of faults. Fault-tolerance is crucial for space systems. There are two types of failures that need to be taken into account for SB-WSNs: The failure of individual nodes and the loss of data during communication.

In the design of MISA two measures are used to meet the fault-tolerance requirement for SB-WSNs. Firstly, a publish/subscribe mechanism is chosen to act as a central component within the middleware design. Publish/subscribe data centric architectures are naturally fault-tolerant \[187\]. This is because the focus in the system lies in the data itself and not in the nodes. Secondly, a periodical data checkpointing technique is applied to tolerate faults in the network. The checkpointed data is used to recover lost data at satellite node failure.

(4) **Resource awareness**

Most research in WSNs centres on managing the highly limited resources of the nodes in the network \[18] \[148\]. Resource management is a key concern in SB-WSNs. Middleware should provide mechanisms to achieve adaptive and efficient utilization of system resources. System resources in case of SB-WSNs include energy, memory, fuel and processing power. MISA achieves resource awareness through the following,

- A resource management module collects information about the resource availability and reacts appropriately in order to ensure the optimization of the resource consumption.

- The data-centricity approach of the design allows the in-network aggregation of messages and as communications is a primary source of energy consumption, energy savings was achieved.
(5) Mobility support

The mobile nature of SB-WSNs has been explained in section 5.3.3. This mobility affects the design of the publish/subscribe system. MISA implements the enhanced Publish-subscribe system described in section 5.4 in order to provide appropriate mobility support for SB-WSNs.

(6) Scalability

Future SB-WSNs would include thousands of nodes. Therefore middleware for space purposes needs to maintain appropriate performance levels node numbers ranging from tens to thousands of nodes.

6.2.2 The Architectural Design

The logical layering of the architecture of the proposed middleware is shown in figure 6.3. The architecture is divided into the following modules.

- **Publish/subscribe layer** - The publish/subscribe layer is the backbone of the MISA middleware. It is responsible for all the receiving and sending the data and interest packets from and to the network. It also plays a central role in mediating the communications between the middleware modules.

- **Resource-management layer** - The resource management layer commands the allocation and adaptation of resources, so that the QoS requirements specified by the applications can be met.

- **Services layer** - The services layer is included in the architecture in order to demonstrate the addition of services to the middleware design. The additional services maybe easily integrated to the middleware if they implement the appropriate interfaces. An aggregation service is included for demonstration purposes.

6.3 Middleware Implementation

6.3.1 The Execution Environment

Before going into the details of the implementation it is necessary to provide an overview of the NesC language that is used for the implementation of the middleware [103]. NesC is a C-like programming language for WSNs. In fact NesC is an extension of the C language,
which makes it a superset of the C language. Key aspects of NesC are an event-driven execution model and the support of a component-based application design.

The basic concepts of NesC are:

- **Component** – Components are the building blocks of applications. Components consist of two parts: a specification and its implementation. The specification is a set of Interfaces whereas the implementation can either be a Configuration or a Module. Building components and wiring them together form applications.

- **Interface** – Interfaces specify a set of Commands that the interface provider has to implement as well as a set of Events that must be implemented by the interface user.

- **Module** – Modules implement commands of the interfaces that it provides. Modules can also use other interfaces and therefore it both provides and requires a set of interfaces at the same time. Modules may also include ordinary C-functions.

- **Configuration** – A configuration is a high-level component that wires together a set of Modules or other nested Configurations. A configuration does also provide as well as require a set of interfaces. It is possible that a configuration has the same interface listing and the “uses” and “provides” clause as with a module. This is called Interface Forwarding.
Command – Commands are functions defined in Interfaces in order to be called by the user. Commands are in general non-blocking, i.e., they immediately return to the caller. The caller is passed the results or the termination confirmation and by signaling an Event, similar to a callback in C. Every command has a return type result_t that immediately indicates success (SUCCESS) or failure (FAIL) of the call on invalid arguments in a send command.

Event - Events essentially are callbacks that must be implemented by the interface user.

Task – Tasks form an independent unit of control in TinyOS and are specified by a function defined as task. Tasks are started by a call to the task function using the keyword post, which returns immediately. In TinyOS tasks always run to completion and may only be interrupted by interrupt service routines.

NesC applications are stored in files of the different file types. Assuming a component provides an interface called Foo, then coding conventions applied throughout TinyOS require it to be stored in a file called Foo.nc. The implementation of the Foo interface would then be stored in file FooM.nc. FooM could use another interface called Loo implemented in another module called LooM. LooM can be wired together in a configuration FooC and stored in a file FooC.nc.

6.3.2 Implementation Architecture

In this subsection an overview of the internal details of the MISA middleware architecture is given. It is important to note that our focus is on the architecture itself and not on the individual subsystems. The implementation has two types of design components: interfaces and modules. The modules and interfaces interconnecting them are shown in figure 6.4.

The Interfaces that are used in the design are the following:

Application Programmer Interface (API)

The API interface is provided by the MISA component (the configuration component that combines all the other subcomponents of the middleware) and is used by the Application component i.e. AppM.nc.
interface API {
    command result_t subscribe(InterestArray);
    command result_t publish(DataArray);
    command result_t setAggregation();
    command result_t setCheckpoint();
    event result_t RcvInterest(result_t success);
    event result_t RcvData(result_t success);
}

Publish Subscribe Interface (PubSubI)

The PubSubI interface connects the MisaControl and the PubSub modules.

interface PubSubI {
    command result_t SendPublication();
    command result_t SendInterest();
    event result_t RcvData(DataMsg);
    event result_t RcvInterest(InterestMsg);
}

Real-Time Interface (RTI)

This is the interface between the MisaControl module and the RT Module.

interface RTI {
    command result_t Admit(InterestMsg);
}
Fault-Tolerance Interface (FTI)

This is the interface between the MisaControl Module and the FT module responsible for checkpointing.

```java
interface FTI {
    command result_t Checkpoint(Data);
    event result_t CheckpointOK(result_t success);
}
```

Aggregation Interface (AggregateI)

This is the interface between the Aggregation module and the MisaControl Module.

```java
interface AggregateI {
    command AggrAttributes aggregate(DataMsg, AggFunction);
    event result_t aggrDone(result_t success);
}
```

The modules that constitute the components of the MISA middleware are as follows:

**MisaControl Module**

The state machine that controls the middleware behaviour consists of 5 states: Idle, RT, FT, RM and PubSub. Initially MISA is in an idle state. The state diagram of the controller is shown in figure 6.5. When the application signals that it requires a subscription or publication to be made the state of the middleware becomes RT. While in the RT state the controller signals to the RT module to analyze possibility of accepting the subscription request based on the deadline Qos provided by the application.

**PubSub Module**

The central component in MISA is the publish/subscribe module. This module mediates the communication between the middleware modules and interacts with the operating system and communications stack. The algorithm used for this module was explained in section 5.3.

**Aggregation Module**

The main function of the aggregation module is to reduce the number of messages flowing through the network, using aggregation techniques. There are two essential sub-functions of the module:

- To implement the aggregation points placement algorithm.
- To apply the aggregation function on the messages received and sent by the node.
Real-Time (RT) Module

The RT module is responsible for ensuring that any publication or subscription made by the application would meet the real-time requirements identified by the application. In other words, this module acts as an admissions controller.

Fault-Tolerance (FT) Module

This module implements checkpoint placement algorithms. It also caches messages in case the node is a checkpoint node.

6.3.3 Middleware Operation

The middleware is currently designed to deal with the data transfer scenarios which are explained in this section. The middleware is extensible to include additional scenarios, in particular setting aggregation points and checkpoints using the setAggregation() and setCheckpoint() commands that is provided in the API. Aggregation points are nodes that are selected as a result of a certain algorithm by the middleware as a point of executing an aggregation function on the data that arrives to it. Checkpoints are nodes that are identified
by an algorithm used by the middleware to perform the checkpointing of the data flowing in the network through it, in preparation to replacing the sink node in case it fails.

The operation of the middleware with regards to the data transfer is explained in the series of UML [186] interaction diagrams 6.6 to 6.10. The notations used in the figures are as follows:

- The objects involved in the interaction are arranged horizontally with a vertical line linked to each object.
- Time is represented vertically so that time progresses down the dashed vertical lines.
- Labelled arrows linking the vertical lines represent interactions between objects. These are not data flows but represent messages or events that are fundamental to the interaction.

In figure 6.6, the application layer in the sink node calls subscribe(), which initiates the subscription process. The RT module receives the subscription message and assesses the ability to fulfil the real-time requirements associated with the call. After admission, the interests are passed to the MisaControl module which in turn passes on the interests to the PubSub module. The PubSub module encloses the interest in a InterestMsg packet and sends it down the operating system.

![Interaction Diagram for sending a subscription (Interest) message](image)

Each intermediate node caches the interest through its PubSub module and then forwards the Interest Packet, after updating it, to the network (figure 6.7).
In case that the interest message arrives at a source node, the PubSub module extracts the interest attributes from the packet and passes it to the MisaControl module, which in turn passes it to the application. The Application then calls the publish(Data) procedure which causes a data packet to be injected into the network (figure 6.8).

When intermediate nodes receive the data packet, the MisaControl goes through the state machine that is shown in figure 6.5. Figure 6.9 assumes that the intermediate node is both a checkpoint node and an aggregation point. The data packet is then updated and forwarded to the next nodes in the network.
Aoolication FiT Aaareoation

ReceivedData(Data)
Checkpoint(Data)
CheckpointOK
Aggregate(Data)
AggregateOK(AggData)
sendData(AggData)

Figure 6.9: Interaction Diagram for receiving a data message by an intermediate node

Figure 6.10 shows the interaction diagram of the case when the data message arrives at the sink node. The data attributes are extracted from the data packet by the PubSub module and then passed to the MisaControl and finally passed on to the application.

Figure 6.10: Interaction Diagram for receiving a data message by the sink node

6.4 Demonstration of Middleware Operation

The evaluation of middleware performance is therefore not a straightforward issue and is a research topic on its own [189]. There are several approaches that have been used in evaluating the functionality of middleware. The most commonly used one is by implementing real application scenarios using the middleware and observing the functionality of the middleware at run-time. This is the approach used for evaluating the ObjectAgent Middleware [190]. Performance evaluation of has been extensively carried out on the TAO middleware [191], but this is beyond the scope of this research work. The approach followed for the MISA middleware was to test the simple functionality of the
middleware using the scenario approach and leave the performance evaluation of the middleware (similar to the TAO middleware evaluation) for future work.

The initial testing of the middleware was done on hardware using the following experimental scenario. The hardware setup involved three MICA2 motes: labelled motes 0, 1 and 2. Mote 0 acts as a sink and is connected to a Linux machine using a serial link. Mote 1 is placed in between motes 0 and 2 such that a multi-hop network is formed. Motes 1 and 2 are attached to a sensor board that has a light sensor. The light sensor is used to emulate fuel onboard SB-WSN motes. Mote 0 is programmed to subscribe to “Fuel>5”. The level of light on the nodes 1 and 2 is varied. The output of the LEDs on mote 0 is varied to reflect the value of the average fuel date received from the network. Through this experimental setup the MISA middleware operation was verified.

As only 3 motes were available for the project, the additional testing of the middleware was made by implementing the scenario explained below using the TOSSIM simulator on a Linux machine along with a GUI tool called TinyViz. The experimental scenario was followed and proper operation of the middleware was observed using the DBG=Usr1 command of TOSSIM.

The scenario is made up of a SB-WSN that consists of 8 nodes. One of the nodes (Node 0) serves as the master of the network whereas all the other nodes serve as slaves. The interaction between the master and slave (ignoring the intermediate node for simplicity) is shown in figure 6.11. Initially the master (set as node 0) subscribes to level of ions detected > 1000 units. Three nodes (7, 5 and 3) are preset to detect level of ions > 1000 units. These three nodes publish their data associated with NoOfIons to the network. The published data (packets) arrive at the master, which then decides based on the data that arrived to cause a reconfiguration manoeuvre in order to rearrange the formation for better detection. The master sends a subscribe message with “fuel > 5 units” attribute. Slaves having fuel more than 5 units publish (4, 5 and 6) the value of their remaining fuel: “fuel = x”. The data packets follow the routes that are constructed by the gradients to reach the master node (Node 0).
A snapshot of the GUI during the simulation is shown in figure 6.12. The API maybe used in a MISA program as shown in the pseudo code below. The code shows that the application program consists of a switch statement, which has three main cases that represent the mode of the SB-WSN: Deployment, Science and Reconfiguration. The SB-WSN starts at the deployment phase where the network is stabilized. When the SB-WSN is at a stage were formation keeping is possible, the network moves to the science phase where science data is collected and either utilized on board or sent to the ground. The reconfiguration phase is initiated whenever the ground or a mother satellite decides that there is a need to reconfigure the formation spatially.

```
Switch (Status)
    Case Deployment
        SetCheckpoint()
        SetAggregation()
    Case Science
        Subscribe(NoOfIons > 1000)
        If receivedInterestMsg (scienceData)
            Publish(scienceData)
    Case Reconfiguration
        If (node = leader)
            Subscribe (fuel >5)
        Else
            If (Received InterestMsg(fuel > 5) && (Fuel in node > 5))
                Publish (Fuel)
```
6.5 Conclusions

This chapter provided a description of the design and implementation of a middleware software layer specifically designed for SB-WSNs. The chapter identifies the requirements that middleware for SB-WSN need to satisfy. This includes scalability, modularity, real-time support, fault-tolerance and mobility support. MISA was designed to meet these requirements. While MISA does not propose a particular algorithm for real-time or fault-tolerance, it provides the framework that allows incorporating such algorithms as future work. It provides a framework that includes the basic modular components that allow the requirements to be met in addition to allowing additional modules to be included for missions that have additional requirements.

The middleware design simplifies the design of SB-WSN software and integrates it with the module-based design philosophy for SB-WSN that was proposed in the previous chapters. Compared to the other middleware software that have been mentioned in the first section of the chapter, the MISA middleware has been designed to incorporate the features that satisfy all the requirements of SB-WSN, while the other software focuses on certain aspects of SB-WSN. A3M was designed for distributed computing onboard a single
satellite while MISA is specifically designed for distributed computing over multiple satellites. ObjectAgent does not suit the module-based design that is proposed in this thesis for the design of SB-WSN. SOIS is a middleware for simplifying interfacing between the various devices on board a single satellite. While some middleware software designed for terrestrial WSN applications do put into consideration certain aspects that are required for space middleware, none of them combine all of those aspects in a single design as is the case with MISA.

Most of the components of the middleware were implemented. Certain components are left for future work. Components to be completed include mainly the Fault-Tolerance component responsible for setting up checkpoints. The aggregation component was implemented but requires some further development to be fully functional as required. The operation of the completed software was verified using both hardware and the TOSSIM simulator. While this is a preliminary verification of functionality of the middleware, we believe that the tests are satisfactory given the resources and time available for the project. Further performance testing and verification is recommended as future work.
Chapter 7

7 Conclusions and Future Work

This chapter summarizes the work presented throughout this thesis and draws conclusions providing certain directions to future work.

7.1 Research Summary

This thesis presents research work on SB-WSNs undertaken at the Surrey Space Centre. The nature of this research is complex, multidisciplinary and demanding a whole variety of skills and knowledge in several different areas. The motivation lies in the need of new design methodologies and architectures for formation-flying missions that consist of ultra-small satellite nodes. Research in the area of formation flying missions have focused on important topics such as formation flying control, micropropulsion and intersatellite communications. This main idea that this research work has emerged from is a novel aspect of formation-flying missions that has not been previously considered i.e. distributed computing for formation flying missions. The research on this topic is a natural extension to the previous research on other topics within the formation flying field. This is because formation flying missions are distributed by nature and combine both computing and communications on-board.

Several different fields of distributed computing exist including enterprise distributed computing, cluster computing and recently WSNs. Distributed computing for formation flying missions was found to naturally match WSNs in terms of architecture and functionality. This match lead to the proposal of treating formation-flying missions that consist of ultra-small satellite nodes as a unique type of WSNs termed “Space-Based Wireless Sensor Networks”.

The literature review surveyed both areas of interest for the thesis, WSNs and formation flying missions, separately. It showed that the general architecture of WSNs resembles the architecture of formation flying missions.
Chapter 7: Conclusions and Future Work

The approach that was taken to verify the applicability of the SB-WSN concept was to explore their design space. The dimensions of the design space include mobility, deployment, node resources, communication modality, heterogeneity, infrastructure, network topology, coverage, connectivity, network size and lifetime. The outcome of the study resulted in the identification of the following key differences between ‘mainstream’ WSNs and SB-WSN. The key differences are as follows:

- Mobility: SB-WSN are mobile and the pattern of their mobility is predictable
- Node Design: The design of SB-WSN nodes are far more complicated compared with the mainstream WSN.
- Resource Limitations: The resource limitations of SB-WSN are different than in the case with most WSN applications. In the case of SB-WSN the limitation lies primarily in the remaining fuel and not in the power as is the case of mainstream WSN.

Given the above findings, it was concluded that restructuring the design of SB-WSN is required at both, the network-level and the node-level. Considering the node-level case, a novel module-based methodology was proposed for designing the nodes of SB-WSN. The modular approach focuses on the inter-operability between the modules leading to the possibility of reusing hardware and software module designs. The advantage of the module-based approach over the single module spacecraft design approach is that it provides a suitable trade-off between flexibility and low-cost. The architecture of hardware modules consists basically of a Multi-Chip Module (MCM) that incorporates at its heart a high-density FPGA that offers flexibility and possibility of incorporating intelligence into the module. The design also incorporates a SpaceWire Codec that is used for implementing an on-board data network between the modules. SpaceWire is therefore adopted as the data interface for the hardware modules that enable inter-operability between these modules.

In order to provide a demonstration to designing hardware modules mentioned above, an on-Board Computer hardware module was designed. Previous work has covered this area particularly in terms of System-on-Chip design, however this is the first research work that has proposed the integration of System-on-Chip and Multichip-Module design for the purpose of designing complete spacecraft systems. The design incorporates 384 MB of SDRAM while the RAD-750 OBC, for example, has 128MB of SDRAM. However, while the designed module meets the mass requirements of on-board computer modules for SB-WSN nodes it still does not meet the projected power requirements.

The OBC Module is an integrated part of a complete distributed system. There are two types of distributed systems that the OBC needs to interact with: the intra-satellite network
and the inter-satellite network. The SpaceWire interface is the window via which the module interacts with the intra-satellite network that in turn, links it to the inter-satellite distributed system.

Regarding the network-level design of SB-WSN, an important design decision that needs to be made is the communication paradigm between the nodes. A data-centric approach is proposed. Using data as the central object of the network rather than the traditional approach of using the nodes themselves incurs several advantages for WSNs (including SB-WSNs) of which the most important is the ability to do in-network processing that result in the optimization of bandwidth usage and therefore significant power savings. Simulations were completed that showed that the mobility of the nodes has a crucial effect on deploying a data-centric network onboard SB-WSN. A novel data-centric protocol was proposed to overcome this hurdle. A purpose built simulator was used to test the functionality of the proposed data-centric mechanism. The simulation results showed that the data-centric mechanism is operational and overcomes the mobility problem that is associated with SB-WSN.

A middleware layer was designed particularly targeting SB-WSN. The middleware was designed putting into consideration the special needs and requirements of SB-WSN. In this research work, the middleware layer design serves the following purposes:

- Provides a demonstration for the design of software part of the proposed module-based design methodology for SB-WSN.
- Provides a demonstration of the implementation of the date-centric network design.

The requirements that middleware for SB-WSN need to satisfy were identified. These include scalability, modularity, real-time support, fault-tolerance and mobility support. MISA was designed to meet these requirements. It provides a framework that includes the basic modular components allowing the requirements to be met. Additional modules could be included too (such as location-awareness mechanisms) for missions that have additional requirements. The component-based architecture adopted by the middleware design integrates it with the proposed module-based design philosophy for SB-WSN.

Compared to the other existing middleware software, the MISA middleware has been designed to provide a framework for incorporating the features that satisfy all the requirements of SB-WSN, while the other software focus on certain aspects of SB-WSN. A3M was designed for distributed computing onboard a single satellite while MISA is specifically designed for distributed computing over multiple satellites. ObjectAgent does not suite the module-based design that is proposed in this thesis for the design of SB-WSN. SOIS is a middleware for simplifying interfacing between the various devices on board a
single satellite. While some middleware software designed for terrestrial WSN applications do put into consideration certain aspects that are required for space middleware, none of them combine all of those aspects in a single design as is the case with MISA.

7.2 Research Contributions

The contributions that have been achieved by this thesis are summarized below:

- **Investigation of the design space of SB-WSN.**

  While formation-flying missions have always been considered as a control problem, we believe that this is the first piece of work that deals with formation-flying from a data processing point of view. Building on that, we also believe that our approach of basing the design of such missions on the use of WSNs technology which naturally emphasizes data collection as the focal design element is a novel approach.

- **Proposal of a module-based design methodology for designing the nodes of SB-WSN.**

  The thesis proposes a new methodology to designing nodes for SB-WSN (formation-flying missions consisting of ultra-small satellite nodes). This approach is different compared with the two concepts that have been proposed so far: SSTL's stack-based approach and the satellite-on-chip approach. The proposed module-based approach is characterized by providing a combination of flexibility and miniaturization to the node design.

- **Design of a miniature OBC module.**

  The thesis presented the design of an OBC Module as part of a module-based design framework for SB-WSN nodes. Previous work has covered this area particularly in terms of System-on-Chip design, however this is the first research work that has covered the integration of SoC and MCM design for the purpose of designing complete spacecraft systems.

- **Evaluation of using the data-centric paradigm for SB-WSN design.**

  This is the first research work that has attempted the comparative evaluation of using data-centricity for space in general and for SB-WSN in particular. The data-centric paradigm was compared against the address-centric paradigm that is currently being adopted by almost all space systems that use networking onboard. The evaluation also showed that the data-centric approach has a serious deficiency due to the mobility of the nodes of the SB-WSN.
• Design and evaluation of a novel data-centric mechanism for SB-WSN.

A novel data-centric protocol was developed as an outcome of evaluating data-centricity for space. The protocol is unique as it is the first mechanism to be developed taking into account the mobility of formation-flying missions as a key design factor. The protocol was evaluated using a purpose-built simulator using open-source software.

• Design of novel middleware architecture to demonstrate the design of software modules and the practical implementation of data-centricity for SB-WSN.

This thesis provides a novel middleware (called MISA) design that was developed specifically for use on board SB-WSNs. Previous work has been attempted on designing middleware for space usage, however the design of MISA targets a special class of missions that has not been addressed before.

7.3 Future Work

While the thesis touched upon several key issues that are essential to the design of SB-WSN several research issues still remain that are important for optimizing their design using the approach that is proposed by this thesis.

7.3.1 SB-WSN

• Development of a distributed computing test-bench for distributed spacecraft missions consisting of multi-robot systems. This would provide a practical methodology to evaluate distributed computing onboard distributed spacecraft missions.

• Deploying SB-WSN consisting of several ultra-small nodes in space. This is the ultimate aim of this research and is a long term objective.

7.3.2 Hardware Module Design

• An on-board computer hardware module was designed in demonstration of the design of hardware modules for the proposed module-based methodology to SB-WSN. However, chapter 3 mentioned that there are two types of hardware modules, wafers that are type I that incorporate a sparse number of electronic devices, numerous micro-channels, plus MEMS and MOEMS; and Wafers that are of type II - wafers that are essentially MCMs that contain most of the centralized signal
processing, command and control electronics and the RF communications. Further work in this area may include demonstrating designing hardware modules that are of type II such as an Attitude Determination and Control (ADCS) module and a navigation module. An ADCS module maybe of particular interest as it would provide a demonstration into the inclusion of MEMS components (such as accelerometers) as part of the module. This would be a step further towards designing complete nodes using the module-based approach.

- Design of an integrated wireless module. The module would include a LEON processor, integrated with SpaceWire, and a wireless physical layer into a single module. TinyOS would need to be ported to the LEON processor and a driver written for the SpaceWire core.

- Extending to the basic design of the OBC module. This extension comes from the need of additional computing power requirements that are expected for future SB-WSN nodes. We have therefore explored the possibility of enhancing the computational power of the OBC module with minimal overhead on the hardware design of the original system. Our proposal is based on including a Multiprocessor system in the design of the system-on-chip as shown in figure 7.1.

![Figure 7.1: Proposed extension to the SoC of the OBC module.](image)

### 7.3.3 Data-Centric Protocol for SB-WSN

- Further evaluation of the protocol: While the data-centric protocol designs for SB-WSN have been designed and evaluated, further evaluations could be completed. The effect of the number of publications and the intervals
between the publications on the performance of the protocol could be analysed.

- Further development in the data-centric protocol: The designed protocol may be developed further to overcome some of the deficiencies and to improve the performance of the protocol. A deficiency in the protocol arises in the case where there are several publications to be made per subscription. This is because this situation invalidates the advantages of the mechanism. A solution for this problem may be taken as further work on the protocol design.

7.3.4 MISA Middleware

Future work on the middleware design includes the following:

- Extensive performance testing of the MISA middleware. Performance measures to be evaluated are throughput, latency and effect of operating system on the performance.

- Incorporating a state-based system to the middleware similar to that introduced by the MILAN middleware [178].

- Currently the API of the middleware is designed for inter-satellite purposes only. Future work includes the incorporation of design considerations for intra-satellite purposes (for example for communications with other modules onboard the satellite node).

- Adding a Resource management module. The resource management module will enable the efficient networking between the nodes in order to enhance their knowledge on the availability of resources. This will enhance the ability of the nodes in making routing decisions leading to greater efficiency within the network.

- The current API design puts Check pointing and Aggregation mechanisms into consideration but the middleware does not implement any particular algorithms. Algorithms could be designed and incorporated in the respective modules within the middleware architecture.

- Integration with agent framework. Autonomy is currently seen as a key technology for future satellite missions. Agent technology is a key part of autonomy research. Integration of MISA with Agents is possible by using agents in the form of higher level components that interact with each other using an Agent Communications
Language (ACL) such as the Foundation for Intelligent Physical Agents (FIPA) standardized messages [192] and interaction protocols [193].
Appendix – A: Miniaturization of Satellite Components and Systems

A.1 Enabling Technologies

Relatively new technologies have been identified as possible methods in achieving satellite miniaturisation. This appendix gives a brief overview of some of these technologies.

A.1.1 MicroElectroMechanical Systems (MEMS)

This is a particularly important technology for spacecraft systems miniaturisation mainly owing to the electromechanical nature of satellites. MEMS technology enables the incorporation of electrical and mechanical systems into a single silicon module.

A.1.2 MultiChip Modules (MCM)

Electronic packaging can count for up to 30% of the total spacecraft mass [195]. Advanced packaging techniques are therefore expected to play an important role in satellite miniaturisation. MCM is one of the state-of-the-art packaging techniques. MCM combine more than one VLSI component onto a unifying substrate or package that usually has some nontrivial interconnection within the substrate between the components. An MCM, besides components, consists of an interconnection substrate, a mechanical substrate, and a package. The package contains a body and I/O terminals for contacting to the MCM substrate on the inside, and the terminals extend to form leads on the exterior of the package to form the next-level interconnect. 3-D MCM can be constructed by stacking the modules vertically as shown in figure A.1 to achieve large volume reductions [115].

Fig A.1: 3-D Multi-Chip Modules (MCM) [115]
A.1.3 Very Large Scale Integration (VLSI):

Moore's law states that the number of transistors per unit area of semiconductor doubles every year [196]. This has a direct effect on the developments of application specific integrated circuits (ASIC) that are becoming denser than before. Field Programmable Gate Arrays (FPGA) is a type of ASICs that offers low-cost and fast prototyping [197]. FPGAs can now be commercially found with densities of up to 8 million system gates [121]. System-on-chip is the integration of various cores (hard or soft) into a single chip. A soft core consists of a synthesizable HDL (Hardware Description Language) description that can be retargeted to different semiconductor processes. A hard core includes layout and technology-dependent timing information and is ready to be dropped into a system [198].

A.2 Miniaturised subsystems

It is evident that in order to reduce satellite mass and volume miniaturisation of the satellite subsystems is essential as it is the subsystems that constitute the satellite. The basic subsystems of satellites are the Attitude Determination and Control (ADCS), thermal system, telemetry and telecommand and power. Propulsion system and navigation are usually optional in most single spacecraft missions. However, for multi-satellite systems flying in formations propulsion and navigation systems are necessary. Recent efforts in the miniaturisation of satellite subsystems are given in this section.

A.2.1 Propulsion

Formation keeping (between the nodes) and reconfigurability are the factors that derive the need of micropropulsion systems onboard of the nodes of virtual satellites. However, the propulsion systems onboard these nodes have different requirements from those currently used onboard of microsatellites. The reason for this is the precise level of actuation required, not only due to formation keeping but also because of the miniature size of the SB-WSN nodes [199].

Virtual Satellite node propulsion systems will have stringent requirements. These propulsion systems have to be physically small and have extremely low mass. As a general rule of thumb the total wet mass of the propulsion systems will form 10-20% of the total node mass [200]. Therefore for a 100grams node, the propulsion system will weigh between 10grams and 20grams. The miniature size of the satellite nodes have a direct effect on the total power they have and therefore the amount of power that is available to the propulsion system. The amount of power that is estimated to be available to the propulsion
system is roughly 1W per Kg of the total node mass [201]. The total power available for a propulsion system of a 100grams node will therefore be 0.1W.

The three main parameters that need to be considered in specifying the requirements of a propulsion system are the thrust (T), the minimum impulse bit (I_{min}) and the specific impulse (I_{sp}). The amount of thrust required depends on the application the thrusters are used for (for example the collision avoidance requires different thrust from formation keeping). In general 1Kg class satellites require 1-1000µN thrusters for typical operations such as orbit maintenance [202]. The required minimum impulse bit is expected to range between $10^4$ and $<10^{-6}$ Ns [62]. High I_{sp} values are desirable to achieve minimum propellant mass.

The vision that has been established in the propulsion community to achieve the above requirements for future virtual satellites propulsion systems is through the MEMS technology described earlier in this chapter. This is clearly evident from the literature that has been published on the miniaturisation of propulsion systems. One of the prominent projects in this area is the digital micropropulsion project [62]. The concept behind digital propulsion is to have an array of one-shot microthrusters fabricated in a three-layer configuration. The three-layer sandwich consists of a top silicon wafer containing burst diaphragms, a middle glass layer containing propellant chambers, and a bottom silicon wafer containing initiators. The prototype system has 19 thrusters on a single chip. Another similar project is the Mega-pixel thruster arrays from Honeywell Technology Center [63]. This is a MEMS thruster array containing a quarter of a million separate thrusters in a small silicon die. Each thruster is addressable and ignitable. Figure A.1.2 shows pictures of the two systems. Table A-1 shows the important parameters that could be obtained for the two projects.

<table>
<thead>
<tr>
<th>Project</th>
<th>Mass (Grams)</th>
<th>Power (mW)</th>
<th>Thrust (N)</th>
<th>Min. Impulse bit (µNs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Propulsion</td>
<td>-</td>
<td>-</td>
<td>0.1</td>
<td>100</td>
</tr>
<tr>
<td>Honeywell Mega-pixel thruster</td>
<td>2.4(including fuel)</td>
<td>10/pixel</td>
<td>-</td>
<td>0.5-20/pixel</td>
</tr>
</tbody>
</table>
A.2.2 Attitude Determination and control (ADCS)

ADCS stabilises the vehicle and orients it in desired directions during the mission despite the external disturbance torques acting on it. It consists of 2 parts, an attitude determination systems (sensors) and a control system (actuators). A lot of research is being carried on MEMS micro-motors that can form the basis of future momentum wheels. However, the lack of very low friction bearing surfaces make this scheme unworkable if not approached from a different angle. The introduction of High-Temperature Superconductors into the design of micro-wheels not only produces almost frictionless magnetic bearings but it also allows the system to be used in energy storage i.e. as a battery [203].

The University of Washington has developed a MEMS Control Moment Gyroscope (CMG) [204]. CMGs consist of spinning disk mounted on gimbals. As the platform is actuated, force is imparted due to the change in direction of the angular momentum vector. Even though MEMS devices can only generate relatively small momentum due to their small size and low weight, arrays of devices and additional miniature flywheels are options to overcome these limitations.

ADCS sensors can be divided into two categories: reference sensors and inertial sensors. Reference sensors give a definite ‘fix’ by measuring the direction of an object such as sun or a star, but there are normally periods of eclipse during which this information is not available. Inertial sensors (such as accelerometers, gyroscopes etc.) measure continuously, but they measure only changes in attitude, they therefore need a fix from reference sensors (such as sun sensors, star sensors and magnetometers).

State-of-the-art miniature sun sensors use either solar cells, a CCD or a CMOS APS for determination of the sun angle. A MEMS sun sensor manufactured by JPL consists of 3
parts: a mask, a spacer and a (CMOS) Active Pixel Array. Images of the sun are formed on the APS image detector when the sun illuminates the mask. Sun angles are derived by determining the precise location of the sun images on the detector. This sensor has a mass of 0.5 grams and the power consumption is 30mW [205]. The Technical University of Denmark for their DTUsat pico-satellite developed a MEMS sun sensor that is based on solar cells that weighs below 3 grams [56].

Star sensors are usually bulky, heavy and require intensive processing and memory resources. The lightest star sensor that could be found weighs 42 grams and consumes 70mW of power [206].

Micro-machined accelerometers operate by monitoring either the motion of a constrained proof mass or the force required to maintain an unconstrained proof mass at a fixed location within the instrument. High sensitivity micro-machined accelerometers such as the silicon electron-tunnelling sensor built at JPL (sensitivity $10^{-7}$g) offer micro-g and better sensitivity for on-orbit applications [207]. Micro-accelerometers with resolution of $10^{-7}$ and better can be used to determine spacecraft orientation by monitoring the gradient of the earth's gravitational field. For launch vehicles and on-orbit propulsion monitoring, micro-machined accelerometers in the range of 1 to 40g can be used. A large number of MEMS accelerometers and gyroscopes are commercially available from a number of manufacturers, including Analog Devices, Kistler, Motorola, Silicon Designs and EG&G IC sensors [77].

Magnetometers are common attitude sensor in attitude determination for the following reasons: they are vector sensors, i.e. they measure both the direction and magnitude of the magnetic field; they require low power for operation and are light weight; they are reliable (up to certain altitudes) and can operate over a wide range of temperatures; finally, they have no moving parts. However, since the magnetic field of the Earth is not precisely known, magnetometers are not accurate. A Tunnelling $\mu$-magnetometer made by JPL offers a unique combination of advantages including high resolution ($\approx 10^{-9}$ Tesla/\sqrt{Hz}), vector sensitivity, wide bandwidth (>10 kHz), low power (<100 mW), small size (1 cm x 1 cm x 0.6 mm), robustness, wide dynamic range (>100 dB), and small temperature coefficient [208].

**A.2.3 Telemetry and Telecommand**

Radio Frequency (RF) systems form a milestone in the area of satellite miniaturisation due to their bulky nature in addition to their power hungry attitude. It was observed (after studying various LEO satellite missions) that Communication systems occupy approximately 10% of the total satellite dry mass. For a 100gram VS node the mass
allocation would be from the range of 8 grams to 15 grams. This is a very stringent requirement given the large distance of transmission between the spacecraft and earth.

Monolithic Microwave Integrated Circuits (MMIC) combines various semiconductor and metallisation layers onto a single chip. MMICs have dramatically reduced the cost and mass of satellite communications systems. However, attempts to further reduce cost by using this miniaturization technique alone have met diminishing returns, indicating that the application of conventional planar technology is reaching its limitations. Hence alternative approaches are required to meet design and performance goals of future systems [195].

Digital modulators and demodulators are being more and more common in satellite communication systems. Digital modulators and demodulators can make use of the advancements in VLSI technologies. Modulators and demodulators can be included as part of the larger system-on-chip including processors, digital filters and other related components and thus achieving miniaturization [209].

Active integrated antenna (AIA) has been a growing area of research in recent years, as MMIC technologies became more mature allowing for high-level integration. From a microwave engineer's viewpoint, an AIA can be regarded as an antenna that can provide certain circuit functions such as resonating, filtering, and diplexing, in addition to its original role as a radiating element. On the other hand, from an antenna designer's point-of-view, the AIA is an antenna that possesses built-in signal- and wave-processing capabilities such as mixing and amplification. A typical AIA consists of active devices such as Gunn diodes or three-terminal devices to form an active circuit, and planar antennas such as dipoles, micro-strip patches, bowties, or slot antennas [210].

Communication circuit miniaturization can be achieved by implementing three-dimensional packaging, where circuits are arranged to be physically interconnected in all dimensions [195]. These approaches integrates diverse technologies, such as High-Frequency electronics, Si-based active circuits, advanced MEMS (e.g. filters) to produce complex packaged systems in a single die would produce high levels of miniaturization. This advanced technology could be applied, for example to a traditional communication transceiver as shown in figure A.1.3 that may use Si/Ge devices, MEMS switches, and micromachined filters and multiplexers [195].
A.2.4 Thermal

Spacecraft thermal control systems can be classified as either active or passive. Passive thermal control techniques consist essentially of the selection of surface properties, control of conduction paths and thermal capacities and the use of insulation systems. Ultra-small satellites will have a larger flux density than the larger micro-satellites. Therefore a thermal system is of critical importance. Active thermal control techniques are in general more complex and heavier than the passive systems and often consume power and telemetry resources. Active thermal-control systems include pumped-loops, heaters controlled by thermostats, and mechanical refrigerators.

Clearly passive thermal control systems are more appropriate for miniaturised satellites due to their lower mass complexity when compared to the active techniques. However active control systems may be required in some cases. Several miniaturised active thermal control system components have been developed. Micromachined heat pipes have been investigated by a number of researchers with some promising results [211].

A micro-machined thermal louver is based on the Texas Instruments Digital Micromirror Device. The vanes and exposed silicon surfaces are coated with vapour-deposited aluminium. When a vane is rotated out of the surface plane, a high emissivity surface of either high or low absorptivity is exposed to the outside environment. Since silicon is transparent to infrared radiation between ~1.2 and 6.5μm, coating would allow a warm object located below the silicon substrate to radiate to space while the vane is open.

Advanced packaging techniques such as MCMs, require a good heat dissipation technique due to the large flux densities. Miniaturised heat pipes that could be used in heat transfer within multichip modules have been prototyped [212]. The addition of micro-heat pipes into micro-channel heat-sink systems allows for the increase in dissipation without increasing the flow-through (pumping power).
A.2.5 Power

The limited surface area of ultra-small satellite nodes limits the total power available for use by the other spacecraft systems. The ever-lasting challenge that is undergone by spacecraft power researchers is to produce solar cells that are smaller in size and mass and higher in efficiency. The limited surface area of the satellite means that only a limited amount of solar power can be converted into electrical energy.

By integrating several different layers of solar-reactive materials such as gallium arsenide, germanium, and silicon into a single cell, researchers have found a means to convert more of the sun's light spectrum into electrical power than state-of-the-art solar cells now in use. The result is more power without increasing spacecraft weight. The latest solar cell technology developed at spectrolab is the improved Triple-Junction cell with a minimum average efficiency of 26.5% developed and manufactured at Spectrolab [213]. Soon, Ultra-Triple-Junction solar cells, with a minimum average efficiency of 28.3%, will enter production.

The current trend for satellite secondary storage is the usage of Li-ion batteries. Li-ion batteries offer high charge densities. Low Earth Orbit satellites undergo over 14 charge/discharge cycles per day, or over 5000 per year. In higher orbits, eclipses are more rare and so a greater depth-of-discharge can be tolerated for the same lifetime. This forms a limitation for the use of Lithium ion cells on-board LEO satellites as the state-of-the-art Lithium ion cells have demonstrated up to 10,000 charge/discharge cycles only. A Lithium micro-battery is under development by JPL that has a feature size of 50 microns. These have excellent cycle life (5000) and deliver 65µA-hr for a typical 1cm² cell [214].

A thin-film solar power system was fabricated that includes both a solar cell and Li-ion batteries in to a single module weighing less that 1.5g with a power output of 170mW. These are particularly interesting because of their capability to be integrated on MCMs or MEMS designed for space applications [215].

Combining components of the power system with components from other subsystems leads to a large reduction in the overall satellite mass. An example of such an approach to minimisation is the attempt to combine solar cells with electronically scannable antennas [216]. The common feature in the operation of antennas and solar cells is that both need a physical aperture. Their electrical performances are proportional to the size of that aperture. The antenna gain increases linearly with its area and so does the electrical power produced by the solar cells. Combining these two apertures into a single one may bring significant advantages in future satellite communications considering their reliability, volume, mass and cost. Another interesting project is the integration of magnet energy
storage system (replacing batteries) with an attitude control system using MEMS technology [217].

A.2.6 Memory

Solid-State memories, being integrated circuits, follows Moore’s law that states that the number of transistors in a chip doubles every 18 months [196]. The current solid-state state-of-the-art for solid-state memories comes in four popular forms: flash cards, compact flash cards, smart media cards and memory sticks. These currently have capacities up to 4 GB for compact flash, 256 MB for smart media cards, 512 MB for memory sticks and up to 1GB for flash cards.

Holographic storage is a new emerging technology with the basic principle involving modulating a light beam representing the data into a 3-D storage medium. Modulating the object beam digitally can be done with a two-dimensional array of optical elements, a task ideally suited to a liquid crystal panel. These are now proposed and used as the spatial light modulators (SLMs) for developmental holographic recording systems. The detector, to suit the two-dimensional nature of the stored data pattern, must be an area detector. A charge-coupled device (CCD) also fits the requirements for a “reader.” The capacity for holographic storage is largely dependent on the medium. However, a CD-sized holographic recording medium is projected to be able to store terabyte capacities [215].

Ferroelectric memory forms a promising future not only to the terrestrial market but also to the space industry due to their use of ferroelectric technology that has already demonstrated fast programming times, fast read times, virtually unlimited endurance, and low programming voltages. Ferro-electric capacitors are tolerant to many forms of radiation. When combined with radiation-hardened CMOS technology, radiation-resistant ferroelectric technology can provide high-speed nonvolatile memories that are suitable for near- and deep-space applications. It has been demonstrated that the ferro-electric process module does not degrade the radiation hardness of underlying CMOS circuitry. Ferroelectric memories can be used in space applications where the low radiation resistance, slow programming times, limited endurance, and high power of other semiconductor nonvolatile memories are unsuitable [219].
Appendix-B: Network Simulation Tools

This appendix gives an overview on the simulation tools that were compared for use in studying data-centricity for SB-WSNs.

OMNET++: The OMNet++ simulator is a public-source simulation environment, which main goal is the simulation of communication networks. It was originally designed for fixed, wired, distributed systems [220]. However its design is quite open, which enables also other target applications [146]. It has a sophisticated Graphical User Interface (GUI) support and commonly used models like IPv4, IPv6, Ethernet etc. are available. OMNet++ is free for academic and non-profit use, however for commercial use a license must be obtained. OMNet++ modules are structured by an own network definition language NED, while the functionality is coded by using C++ classes.

SIMICS: Simics is a full system simulation platform [221]. This means that it simulates everything inside the computer system including the processor, bus interfaces, video cards, disks, etc. The processor, which is the most important part of the simulator, is simulated at the instruction set architecture (ISA) level. This is the lowest level of the computer hardware that software has access to. These properties together make the simulator able to run any software that the target system can, including the operating system, drivers and user code. Despite the fact that the developers of Simics claim that “Simics is highly scalable, supporting simulated systems with hundreds of simulated processors”, the scalability of Simics is dependant on several factors including the ratio of the CPU power of one simulated target to the CPU power of the host system and the RAM size of the simulated target compared to the RAM size of the host system. The level of detail that Simics is capable of simulating does come at the expense of scalability. Another strong downside of Simics is that the creation of new models is a time consuming process. Therefore we have concluded that Simics is an ideal tool for applications that require simulating standalone computer systems (e.g. simulating the behaviour of an on-board computer system in space) but is not suitable for simulating distributed computing for formation flying involving tens or hundreds of satellites.

NS-2: ns-2 is possibly the most prominent network simulator. It includes many protocols, traffic generators, to simulate TCP, routing and multicast protocols over wired and wireless networks [222]. Ns-2 puts much emphasis on following the ISO/OSI model. Ns-2 has a long learning curve and requires advanced skills to perform meaningful and repeatable simulation.
TOSSIM: Tossim is a simulation tool that was developed specifically for wireless sensor network applications [105]. TOSSIM scales to thousands of nodes, and compiles directly from TinyOS code; developers can test not only their algorithms, but also their implementations. TOSSIM simulates the TinyOS network stack at the bit level, allowing experimentation with low-level protocols in addition to top-level application systems. Users can connect to TOSSIM and interact with it using the same tools as one would for a real-world networking, making the transition between the two easy. TOSSIM also has a GUI tool, TinyViz, which can visualize and interact with running simulations. Using a simple plugin model, users can develop new visualizations and interfaces for TinyViz.

EMSTAR: Emstar is a Linux-based software environment for developing and deploying complex wireless sensor network applications on networks that may consist of 32-bit micro-server platforms integrated with networks of motes (8-bit devices) [223]. It provides diverse tools and services that have high potential use.

OMNET++ is selected as the simulation environment to be built on for the purpose of this research for two main reasons. Firstly, OMNET++ can be used to simulate different kinds of modules relatively easily due in C++. Secondly, it provides the appropriate level of abstraction for distributed computing simulations. The main features that OMNET++ possess are:

Its main features are:

- It is a discrete time simulator. The simulated objects communicate with each other by exchanging messages at discrete moments of time;
- It is written in C++ and Tcl/Tk;
- Several graphical interfaces allows easy debugging and variables inspection. It also offers support for recording data vectors and scalars in output files;
- It offers support for parallel execution;
- Simulated objects are represented by modules. The modules can be simple or composed (depth of module nesting is not limited). The modules communicate by messages (sent directly or via gates). Each module description consists of an interface description and a behavior description;
- Several random generators (also from several distributions) with different starting seeds - a quite useful tool for simulating random satellite events that occur in the space environment
Simulations are easy to configure using .ini file. Batch execution of the same simulation for different parameters is also included.
Appendix – C: Derivation of Data-Centricity Equation

The following analysis extracted from reference [145] shows the advantage of using a data-centric approach over the address-centric approach for networking in the context of WSNs in general. Consider the network in figure C.1. The network shows the configuration of a typical SB-WSN with the nodes in orbit and a relatively distant node receiving data from the network (a mother satellite, relay satellite or a ground station).

![Network Model for Evaluating Data-Centricity for SB-WSN](image)

**Figure C.1: Network Model for Evaluating Data-Centricity for SB-WSN**

**Definitions:**

- \( k \) = Number of possible sources in the network.
- \( i = 1, 2, \ldots k \) are integer numbers.
- \( d_i \) = The distance of the shortest path (number of hops) from source \( S_i \) to the sink.
- \( N_A \) = Total number of transmissions required for the address-centric protocol.
- \( N_D \) = Total number of transmissions required for the data-centric protocol.

The total number of transmissions required for the optimal address-centric approach is:

\[
N_A = d_1 + d_2 + \ldots d_k = \sum (d_i)
\]  

(1)
Appendix-C

**Proposition 1:**

\[ N_D \leq N_A \]

**Proof:**

Doing data aggregation optimally can only decrease the minimum number of required communication links needed compared to the situation when sources send information to the sink along shortest paths.

**Definition:** The diameter \( X \) (in number of Hops) of a set of nodes \( S \) is the maximum of the pair wise shortest paths between these nodes,

\[ X = \max_{i,j \in S} SP(i, j) \]

where \( SP(i, j) \) is the shortest number of hops needed to go from node \( i \) to \( j \).

**Proposition 2:**

If the source nodes \( S_1, S_2, \ldots, S_k \) have a diameter \( X \geq 1 \), the total number of transmissions \( (N_D) \) required for the optimal data-centric protocol satisfies the following bounds:

\[
N_D \leq (k - 1)X + \min(d_i) \quad (2) \\
N_D \geq \min(d_i) + (k - 1) \quad (3)
\]

**Proof:**

Expression (5-2) can be obtained by construction of what is called “the data aggregation tree” which consists of \((k - 1)\) sources sending their packets to the remaining source which is nearest to the sink. Expression (5-3) is obtained by considering the case when \( X=1 \), which is the case with the smallest possible tree where there is exactly one connection from each of the other source nodes to this node.

**Proposition 3:**

If the diameter \( X < \min(d_i) \), then \( N_D \leq N_A \). In other words, the optimum data-centric protocol will perform better than the address-centric protocol.

**Proof:**

\[
N_D \leq (k - 1)X + \min(d_i) < (k) \min(d_i) \\
\Rightarrow N_D < \sum(d_i) = N_A
\]

**Definition:** The fractional savings obtained using data-centric protocol as opposed to the address-centric protocol = FS.
Proposition 4:
The fractional savings FS lies in the following bounds:

\[ FS \geq 1 - \frac{(k - 1)X + \min(d_i)}{\sum(d_i)} \]  \hspace{1cm} (6)

\[ FS \leq 1 - \frac{\min(d_i) + k - 1}{\sum(d_i)} \]  \hspace{1cm} (7)

Assuming that all the sources are at the same shortest-path distance from the sink i.e. \( \min(d_i) = \max(d_i) \), then we have that:

\[ 1 - \frac{(k - 1)X + d}{kd} \leq FS \leq 1 - \frac{d + k - 1}{kd} \]  \hspace{1cm} (8)

Proposition 5:
Assuming that \( X \) and \( k \) are fixed, then as \( d \) tends to infinity (i.e. as the sink is farther and farther away from the sources):

\[ \lim_{d \to \infty} FS = 1 - \frac{1}{k} \]  \hspace{1cm} (9)

Proof:
In the limit, \( X \ll d \) and \( k \ll d \). It suffices to show that both lower and upper bounds in equation (8) converge to the same right hand side value:

\[ \lim_{d \to \infty} \left( 1 - \frac{(k - 1)X + d}{kd} \right) \]

\[ = \lim_{d \to \infty} \left( 1 - \frac{(k - 1)X}{kd} - \frac{d}{kd} \right) \]

\[ = 1 - \frac{1}{k} \]

and

\[ \lim_{d \to \infty} \left( 1 - \frac{d + k - 1}{kd} \right) \]

\[ = \lim_{d \to \infty} \left( 1 - \frac{d}{kd} - \frac{k - 1}{kd} \right) \]

\[ = 1 - \frac{1}{k} \]
Bibliography


Benoit Hairabetian, 3D-Plus Inc. (Private Communication), 2005.


Bibliography


[200] A. D. Ketsdever, “System considerations and design options for microspacecraft propulsion systems,” in Micropropulsion for small spacecraft, M. Micci and A.
Ketsdever, [Great Britain]: American Institute of Aeronautics and Astronautics, 2000, Chapter 4, pp. 139-166.


