Adaptability in Wireless Sensor Networks Through Cross-Layer Protocols and Architectures

by

Christophe J. Merlin

Submitted in Partial Fulfillment
of the
Requirements for the Degree
Doctor of Philosophy

Supervised by
Professor Wendi B. Heinzelman
Department of Electrical and Computer Engineering
Arts, Sciences and Engineering
School of Engineering and Applied Sciences

University of Rochester
Rochester, New York

2009

Curriculum Vitae

The author was born in Vendôme, France, on March 1, 1980. He attended the Institut National des Sciences Appliquées de Rennes, France from 1998 to 2003 and graduated with double degrees of Bachelor of Science and Master of Science in Electrical Engineering. He began graduate studies at the University of Rochester in 2004. He has since pursued his research in wireless sensor networks under the supervision of Dr. Wendi Heinzelman and received his Master of Science degree in Electrical Engineering in 2006.

Acknowledgements

The present dissertation is a testimony to Dr. Wendi Heinzelman's invaluable support over the years. I would like to thank her for giving me the opportunity to pursue research in the exciting field of wireless communications.

I would also like to express my thanks to Professors Mark Bocko, Azadeh Vosoughi, and Henry Kautz for acting as members of my thesis committee.

Over the years at the University of Rochester, I have had the pleasure to work with exceptional colleagues in the Wireless Communications and Networking Group. Among them, I would like to specifically acknowledge Mark Perillo, Tolga Numanoglu, and Stanislava Soro for their help and friendship. I have benefited from collaboration with everyone in the laboratory.

I also would like to thank my family and friends, most of all my soon-to-be wife Catherine, for offering encouragement and patience as I finished my degree. I would like to express my gratitude to my parents for their strong values and the courage they showed in supporting my move to America.

This research was made possible financially in part by the National Science Foundation through grant CNS-0448046.

Abstract

Complex applications and increased sensor capabilities will help proliferate wireless sensor networks into everyday life. However, as sensor nodes are battery-operated, sensor networks will require protocols to spare every possible bit of energy. This can be accomplished through cross-layer protocol optimizations that specialize the protocols for specific application requirements. However, as researchers continue to contribute to the field of sensor networks, protocols will evolve, and more efficient work will replace older ideas. Therefore, flexibility and eased maintenance of the network will be required to make sensor networks feasible for new deployments and customers. Thus there are competing goals of energy-efficiency, achieved by specialization through cross-layer protocol design, and flexibility, achieved through modularity in a layered protocol design.

My thesis shows that these competing goals can be balanced by the use of cross-layer information exchange that enables the protocols (and hence the network) to adapt to current application and network conditions. Adapting the protocols via cross-layer information exchange allows the network to make best use of the limited energy resources of the sensor nodes while maintaining required application quality of service and retaining a flexible protocol stack. In support of this thesis, we have (1) performed a case-study comparing cross-layer designs with a layered design, (2) designed and evaluated an architecture that enables the sharing of information across traditional stack boundaries, (3) proposed a solution to manage the bidirectional flow of information between middleware and the protocol stack, (4) demonstrated the advantage of allowing a sensor network middleware to exploit this cross-layer information to maximize the time when application quality of service is met, and (5) developed various techniques at the MAC layer to take advantage of the availability of this cross-layer information to extend node lifetime. With the standardization of cross-layer information exchange

provided by our new architecture, and the many example protocol adaptations provided in this dissertation, one can envision new designs at all protocol levels, making sensor networks truly adaptive to changes in both application requirements and network conditions.

Contents

1 **Introduction** **1**
1.1 Overview of Wireless Sensor Networks 2
1.2 Research Motivation 4
1.3 Research Contributions 5
1.4 Thesis Structure 6

2 **Related Work** **8**
2.1 Layered Protocol Stack 8
2.1.1 Node Activation 8
2.1.2 Routing 9
2.1.3 Medium Access Control 9
2.2 Cross-layer Protocol Stack 12
2.2.1 Cross-Layer Protocol Limitations 13
2.2.2 Cross-Layer Protocols 14
2.3 Adaptability in Wireless Sensor Networks 15
2.4 New Architectures 16
2.5 Services 18
2.6 Middleware for Managing Cross-Layer Protocols 20

3 **Gains and Limitations of Cross-Layer Designs** **22**
3.1 Introduction 22
3.2 DAPR: A Cross-Layer Protocol 23
3.2.1 Route Discovery Phase: Selecting Minimum Application Cost Paths 23
3.2.2 Role Discovery Phase: Providing Maximum Coverage 25

3.2.3 Integration with the MAC and Physical Layers 25
3.3 Simulation Scenarios and Results 26
3.3.1 Methodology: the Discrete and Continuous Cases 26
3.3.2 Continuous Monitoring of a Region 27
3.3.3 Discrete Monitoring of a Region 30
3.3.4 Non-uniform Node Deployment 33
3.4 Gains of Cross-Layering Within the Same Protocol 35
3.4.1 Advantages and Drawbacks 35
3.4.2 Additional Results: the Impacts of Cross-Layering 37
3.5 Discussion of the Usefulness of Cross-Layering 42
3.6 Summary 43

4 A New Cross-Layer Information-Sharing Sensor Network Protocol Architecture 45

4.1 Introduction 45
4.2 Architectural Approaches 48
4.2.1 MobileMan: Subscription to an Abstracted Database 49
4.2.2 CrossTalk: A Common Database for MANET 50
4.2.3 SNA: Abstraction of Lower Layers and Basic Functions 52
4.2.4 Chameleon: Abstraction of Communication Protocols 53
4.2.5 XLM: The Counterpart—Fused Layers 53
4.3 Architecture Comparisons 54
4.3.1 Flexibility 55
4.3.2 Information Freshness 57
4.3.3 Overhead 58
4.3.4 Simplicity 58
4.4 X-Lisa, a New Architecture for Cross-Layer Information Sharing 59
4.4.1 The Need for a New Architecture 59
4.4.2 A New Unifying Architecture 60
4.4.3 Information Sharing Structures 61
4.4.4 Event Signaling 66
4.4.5 Information Exchange 66
4.4.6 Maintenance of the Neighbor Table 68

4.4.7 Important Services 69
4.5 Implementation Details 70
4.5.1 Components in TinyOS 70
4.5.2 Information Storage 72
4.5.3 DAPR and the Proposed Architecture 73
4.5.4 GFG / GPSR and X-Lisa 78
4.6 Results 78
4.6.1 Modus Operandi 80
4.6.2 Qualitative Study: The Expressiveness of X-Lisa 81
4.6.3 Quantitative Study: Measurable Cross-Layer Improvements . . 84
4.7 Discussion 87
4.7.1 X-Lisa Extends the Desirable Properties of Existing Architectures 88
4.7.2 To Use or Not to Use X-Lisa 89
4.7.3 The Size of X-Lisa 90
4.7.4 The Most Gain for the Cost 91
4.8 Summary 91

5 Supporting Proactive Application Event Notification to Improve Sensor Network Performance 93
5.1 Goals and Challenges of WSN Architectures 95
5.2 Middleware Support 96
5.2.1 General Ideas 96
5.2.2 Integration Into an Information-Sharing Architecture 97
5.2.3 Composite Query Registration and Deregistration 100
5.2.4 Interest Registration and Deregistration 101
5.2.5 Query Notification 101
5.3 Evaluation of Middleware Support 103
5.3.1 Middleware Interpreter Only 103
5.3.2 X-Lisa with Middleware Interpreter 108
5.4 Summary 114

6 Middleware for Supporting Protocol Adaptation 117
6.1 Introduction 117

6.2 MiLAN: a Sensor Network Middleware . . . 118
6.3 Managing DAPR Through MiLAN . . . 119
    6.3.1 Overview of MiLAN/ DAPR Combination . . . 120
    6.3.2 Routing: New Variable-based Cost . . . 122
    6.3.3 Node Activation: a Distributed Process . . . 123
6.4 Results . . . 123
    6.4.1 One Variable with Low QoS Requirements . . . 125
    6.4.2 QoS Increase with a Middleware . . . 127
    6.4.3 Change of Query Interval . . . 128
6.5 SRI: A General Cost Function . . . 132
6.6 Summary . . . 132

7 Schedule Adaptation of the Low-Power-Listening MAC Protocol Family 134
7.1 Introduction . . . 135
7.2 MiX-MAC: A Highly Adaptable MAC Protocol . . . 137
    7.2.1 Principles of MiX-MAC . . . 137
    7.2.2 X-MAC: A Short Preamble MAC Protocol . . . 138
    7.2.3 MX-MAC: a *LPL* Variant of CSMA-MPS Compatible with X-MAC and SpeckMAC Schedules . . . 139
    7.2.4 SpeckMAC-D: Repeating the Data Packet . . . 140
    7.2.5 Lifetime Calculation . . . 140
7.3 Fine Tuning Channel Probing Protocols . . . 142
    7.3.1 Adapting the Channel Probing Interval t_i . . . 143
    7.3.2 Adapting the Packet Transmission Schedule . . . 145
7.4 Simulation Comparison of MAC Protocol Scheduling Techniques . . . 146
    7.4.1 Performance Comparison . . . 147
    7.4.2 MiX-MAC: Adapting the MAC Schedule to Conditions in the Network . . . 153
7.5 TinyOS Implementation of *LPL* Protocols . . . 154
    7.5.1 Debugging at the MAC Level . . . 156
    7.5.2 Reconstruction Model . . . 156
    7.5.3 Protocol Design Choices . . . 160
    7.5.4 Determination of the Switching Thresholds . . . 163

7.5.5 Reliable Throughput or Goodput 164
7.5.6 MiX-MAC Achieves the Upper Bound of Node Lifetime 166
7.6 Discussion and Summary 177

8 Node Synchronization Along a Path 181
8.1 Synchronization Over a Unidirectional MX-MAC Link 182
8.1.1 Principle 182
8.1.2 Synchronization Process 184
8.1.3 Urgent Packets 186
8.1.4 Pipelining of Packets on a Synchronized Path 187
8.2 Synchronization Over a Bidirectional Path 187
8.3 Path Synchronization With Multiple Sources 189
8.3.1 Synchronization Over Several Unidirectional Paths and Conflict Resolution 189
8.3.2 Strategy 190
8.4 Simulation and Implementation of Synchronization Principles 192
8.4.1 Simulations 192
8.4.2 Implementation on Tmote Sky 199
8.4.3 Combined Effects of MiX-MAC and Node Synchronization . . 201
8.4.4 Multi-Source Synchronization Implementation 203
8.5 Summary 204

9 Duty Cycle Control for Dynamic Adaptation of Low-Power-Listening MAC Protocols 207
9.1 Estimation and Control for Multi-variable Systems 208
9.1.1 Background 209
9.1.2 The Adaptive Regulator 210
9.1.3 Evaluating the Target Energy 215
9.1.4 Algorithm for t_i Control 216
9.1.5 Preliminary Guidelines for an Implementation 216
9.2 t_i Control For Multi-Hop Networks 219
9.2.1 Challenges Introduced By Multi-Hop Control 219
9.2.2 Node Synchronization Along a Path 220

9.2.3 Impact on the Energy Component of J 221
9.2.4 Practical Considerations 221
9.2.5 Simulation Results 223
9.3 t_i Control for Multi-Hop Networks With Multiple Sources 226
9.3.1 t_i Control For Synchronized Paths With Multiple Sources . . . 226
9.3.2 Simulation Results 227
9.4 Summary 227

10 Conclusions and Future Directions **230**
10.1 Summary of Contributions 230
10.2 Future Directions 231

Bibliography **233**

List of Tables

4.1 Qualitative comparison of existing architectures (given without order of importance). 55

4.2 A neighbor table is kept at every node i with non-predetermined fields. It keeps information about the node itself (for vertical cross-layering) and each of its neighbors j (for horizontal cross-layering). 62

4.3 The default fields of the neighbor table. 63

4.4 A sink table is kept at every node with information about each sink j in the network. 64

4.5 A message pool is kept at every node. 65

4.6 A packet with a *CLOI* information vector piggy-back (TOS_Msg fields not included). 67

4.7 Example of a routing packet definition. The application layer may dispose of the space defined by *data*. The MAC protocol only sends the used bytes (S) in the packet, and discards those that are not in use (X). . . 72

4.8 The MAC protocol only ignores bytes placed at the end of a packet (designated by X). If two fields of varying size are defined, the MAC sends a section filled with unused and unassigned bytes (U). 72

4.9 *CLOI* rearranges bytes in order to place all unused bytes at the end of a packet. The MAC protocol may then discard the unused bytes. 73

4.10 Selected metrics comparing the behaviors of XLM / X-LISA and XLM / LPL / X-LISA. 83

5.1 Some of the fields of the Middleware Interpreter with their TinyOS primitive type and an example. 99

5.2 Composite Query Stored within the Middleware Interpreter 101

5.3 Query Notifications and Status Based on the Preexisting Status of a Query and Whether Its Conditions Are Met Upon an Update in the Neighbor Table 103
5.4 Application requirements for Middleware Interpreter only scenario . . . 105

6.1 Detailed calculation of the application costs for 4 sensors and 2 variables. The final costs reflect that the same importance is granted to a precision of 0.5 and 0.9—the application has no use for a precision higher than 0.5 and lower than 1.0. The tables at the bottom give the values of P and W. 124

7.1 Notations and values for the CC2420 radio. 141
7.2 Time and energy analysis of X-MAC (X) and SpeckMAC-D (S) for unicast transmissions. 142
7.3 CC2420 Radio Parameters 147
7.4 Actual radio model under TinyOS. *S* designates the packet size in bytes. Units are μJ and ms. 159
7.5 Look up table to determine which protocol (SpeckMAC-D (S), X-MAC (X), or MX-MAC (M)) performs best in terms of lifetime. 175

8.1 Energy Consumption and Packet Delay. 198

List of Figures

2.1 (a) Layered structure. (b) Cross-layer structure with fused layers. (c) Cross-layer structure with information exchange between layers. 13

2.2 Time diagram of the message delay estimation. The sender may measure the delay as $(t_{1-2} - t_{1-1}) - (t_{2-2} - t_{2-1})$ or $(t_{1-4} - t_{1-3}) - (t_{2-4} - t_{2-3})$. The receiver can estimate the packet delay as $(t_{2-3} - t_{2-2}) - (t_{1-3} - t_{1-2})$. 19

3.1 (a) DAPR rounds consist of three phases. (b) A sensor checks whether its coverage is overlapping with that of others during the role discovery phase. Sensor 2 is not needed to cover the rectangular area. 24

3.2 Continuous monitoring: network coverage over time for modified DAPR ($C_{ac}(S_i) \propto \frac{1}{\epsilon_{rem}}$) and PEAS, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 100 nodes. DAPR outperforms PEAS, and the contention incurred by 802.11 causes nodes to wrongly decide to sleep in the first moments of the simulation. DAPR+RTdma, as a fully cross-layer design, prevents deactivation mistakes. 28

3.3 Continuous monitoring: network coverage over time for DAPR with its original ($C_{ac}(S_i) \propto \frac{1}{\Sigma\epsilon_{rem}}$) and modified ($C_{ac}(S_i) \propto \frac{1}{\epsilon_{rem}}$) costs, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 100 nodes. With RTdma, the original cost allows for gains of 3% to 15% over the energy cost. 30

3.4 Continuous monitoring: network coverage over time for DAPR with original cost ($C_{ac}(S_i) \propto \frac{1}{\Sigma\epsilon_{rem}}$) and PEAS, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 100 nodes. For this application, PEAS is greatly outperformed, and the MAC level inefficiencies of DAPR are no longer observed. . . . 31

3.5 Discrete monitoring: network coverage over time for DAPR and PEAS, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 100 nodes. For this application, DAPR outperforms PEAS for coverage $> 85\%$ but the low overhead of PEAS guarantees a longer lifetime for mid-to-low range coverage. . . . 32

3.6 d) shows a typical node deployment for discrete monitoring (the inner and outer radii are $70\ m$ and $100\ m$). a), b) and c) are the network coverage over time for 125 nodes for 70%, 80%, and 90% of the nodes deployed in the ring. e) is the difference in lifetime ($Lifetime_{PEAS} - Lifetime_{DAPR}$ in seconds) for various ρ at 90%, 70%, and 50% coverage. As the non-uniformity of the deployment increases, so does DAPR's relative performance. 34

3.7 Continuous monitoring: network coverage over time for cross-layer, hybrid, and layered modified DAPR using the energy cost $\frac{1}{\epsilon_{rem}}$. (b) is the network lifetime for 25, 50, 75, 100, and 125 nodes. (a) is the coverage percentage as a function of time for 125 nodes. The fully cross-layer and hybrid designs perform similarly for high-range coverage, and the only significant difference happens for low-range coverage. 36

3.8 The numbers represent: *NodeId* in circles, *Individual Cost - Cumulative Cost*. Recall that nodes with the smallest cost forward the *query* first. If nodes 3 and 4 have overlapping sensing areas, node 4 (individual cost of 20) will activate but is part of a path with higher cost than 3 (individual cost of 25). Node 7 will thus route its packets through 4, using a less efficient route than if 3 had activated. 37

3.9 Continuous monitoring: network coverage over time for cross-layer, hybrid, and layered DAPR, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 125 nodes. The cross-layer and hybrid show no significant difference in lifetime. 39

3.10 Discrete monitoring: network coverage over time for cross-layer, hybrid, and layered DAPR with original cost, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 125 nodes. No significant difference is observed between cross-layer and hybrid designs. 40

3.11 Continuous monitoring in a non-uniform deployment with 90% of the nodes at the periphery: network coverage over time for cross-layer, hybrid, and layered DAPR, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 125 nodes. No significant lifetime gain is obtained for cross-layer DAPR over the hybrid scheme even though the non-uniformity should advantage the former. 41

4.1 Classification of some architectures for MANET and WSN. MobileMan abstracts a data base through signaling, as highlighted by the dotted circle. 50

4.2 X-Lisa: An information-sharing sensor network architecture for cross-layer optimizations. This architecture retains a layered design while providing flexible information repositories as well as services to support the protocols. 61

4.3 The X-Lisa architecture in TinyOS. The shaded area corresponds to *CLOI*. Implementations are in large characters and delimited by solid line frames. Each component is bounded by a dashed line, and its name appears vertically. 71

4.4 The original XLM (a), was broken into a layered scheme (XLM / X-Lisa) (b), and its MAC layer was replaced (c). Arrows show packet exchanges between layers, and squares information exchange. 82

4.5 Simulated network topology (units in meters). The nominal radio range is $30\ m$. 83

4.6 Comparison of the impact of DAPR and DAPR + X-Lisa on the network for 5 and 10 nodes. The numbers on the graph are the relative change from DAPR to DAPR + X-Lisa. 86
4.7 Comparison of the QoS of DAPR and DAPR + X-Lisa on the network for 5 and 10 nodes. The numbers on the graph are the relative change from DAPR to DAPR + X-Lisa. 87

5.1 New X-Lisa architecture with middleware support. 98
5.2 Event filtering and notification process: 1. A service or protocol updates the neighbor table. 2. If the field is of interest, and the new value is different from the old one, it is submitted to the *MI*. 3. The *MI* checks conditions realizing a query. 4. The *MI* notifies subscribing protocols and services that the query has fired. 102
5.3 Topology of the (a) health monitor, and (b) room monitor test network. . 104
5.4 Comparison of the (a) delays (b) PDR for the patient monitoring scenario with and without the Middleware Interpreter. 107
5.5 Comparison of the number and nature of packets on the network for the low mobility scenario with and without X-Lisa. 110
5.6 Comparison of the (a) delays and (b) packet delivery ratios for the room monitoring scenario with low node mobility with and without X-Lisa. . 112
5.7 Comparison of the number and nature of the packets on the network for the room monitoring scenario with high node mobility with and without X-Lisa. 113
5.8 Comparison of the (a) delays and (b) packet delivery ratios for the room monitoring scenario with node high mobility with and without X-Lisa. . 115

6.1 Overview of MiLAN. A and B start when the application starts or changes its state depending on data received from the sensors. C is the normal mode of operation when conveying the sensors data to the application. 119

6.2 Nine sensors are monitoring two different entities (*e.g.*, two different soldiers in range of one another). The required precisions for the two variables of interest are 1.0 and 0.9, with a tolerance of 0.2. Three cases are possible: only the first entity (with sensors represented by circles), only the second entity (with sensors represented by stars) or both entities are of interest. 125
6.3 Node remaining energy over time. All three nodes are able to monitor a single variable with precisions 0.1, 0.5, and 0.9. The QoS requirement is 0.1. a) and b) have different P vectors. 126
6.4 A measure of the QoS differences between a system with MiLAN managing DAPR and a system designed for the strictest QoS. The graph at the bottom shows the application QoS requirement. 127
6.5 (a) Lifetime of the network as defined by the time when the network was first incapable of providing a coverage of $x_F\%$ for different but fixed values of the query length interval. (b) zooms on the results for high-range coverage. 129
6.6 (a) Lifetime of the network as defined by the time when the network was last capable of providing a coverage of $x_L\%$ for different but fixed values of the query Length interval. (b) zooms on the results for high-range coverage. 130
6.7 Network QoS response of the adaptive (a) and fixed (b) query length schemes to the application needs (c). 131

7.1 MAC schedule for B-MAC, X-MAC, MX-MAC, and SpeckMAC-D. . . 136
7.2 Lifetime as a function of t_i for SpeckMAC-D and X-MAC for various ratios of transmissions vs. receptions. 144
7.3 Adapting the node's schedule to packet types. 146
7.4 Node's lifetime for X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK with a fixed n. All packets sent are (a) broadcast, (b) unicast. . 149
7.5 Node's lifetime for X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK with a fixed n. All packets sent are unicast, and $m = n + 1$. . . 151

7.6 Node's lifetime for X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK with a fixed n. All packets sent are unicast, and the packet size has doubled to 80 bytes. 152
7.7 5 nodes attempt to send 10 packets at a rate $1/5$ for various t_i values. (a) gives the packet delivery ratio, (b) the channel usage, and (c) the average delay. 153
7.8 Node lifetime as a function of t_i for MiX-MAC, X-MAC, MX-MAC and SpeckMAC-D. The packets are 20% broadcast / 80% unicast (a) and all unicast (b), sent at a rate $r = \frac{1}{10}$. 155
7.9 Picture of the acquisition board set up to measure the current drawn from the mote. 157
7.10 Current drawn by the Tmote Sky during a medium probe (a), a $40\ B$ packet transmission (b), and a $40\ B$ packet reception (c). 158
7.11 Relative difference between real life scenarios and their prediction through the reconstruction model. 160
7.12 Probability to successfully hear an ongoing stream of packets as a function of the packet size (L) and t_{2CCAs}, for an RX / TX switch time t_{switch} = (a) $770\ \mu s$ (SpeckMAC), and (b) $1,350\ \mu s$ (X-MAC / MX-MAC). 168
7.13 Comparison of the packet delivery ratio of the MAC schedules as a function of packet size and time between CCAs. 169
7.14 Throughput for t_i values of $250\ ms$, $500\ ms$, and $1\ s$. 170
7.15 Comparison of the lifetime of SpeckMAC and MiX-MAC. The two schedules only differ by the time between their CCAs. In (b) we assume that the receiver notifies the sender of the reception of packets through piggy backed or stand-alone ACKs. (a) assumes no such mechanism. . . 171
7.16 Comparison of the MX-MAC and X-MAC schedules for $15\ B$ unicast packets for scenarios where the node is mostly sending (a), and mostly receiving (b). 172
7.17 Comparison of the MX-MAC and X-MAC schedules for $40\ B$ unicast packets (a) and $100\ B$ packets (b) when the node is mostly sending. . . 173

7.18 Comparison of the MX-MAC and X-MAC schedules for $40\ B$ broadcast packets (a) when the node is mostly receiving, and $15\ B$ packets (b) when the node is mostly sending. 174
7.19 A simple mapping function lets the protocol switch between schedules in order to increase the lifetime. 176

8.1 Synchronization principle for (a) two (b) three nodes running an *int-LPL* protocol. 183
8.2 Node 0 pipelines packets and increases the packet rate. 187
8.3 Bidirectional path synchronization time line. 188
8.4 (a) Synchronized nodes along two parallel paths: nodes $\{10, 11, 12, 30\}$ form one path, and $\{20, 21, 22, 30\}$ another one. The dotted lines indicate that the nodes can communicate with each other (and thus interfere). (b) Mitigation of the problem. 189
8.5 A multi-hop network with two sources $k - 1_0$ and $k - 1_1$. 191
8.6 On the path {1, 5, 4, 3, 10}, the nodes synchronize correctly after only a few packets. 193
8.7 (a) Packet delay on the same path as Figure 8.6. (b) Same configuration, but packets 7 through 15 are marked as urgent. 195
8.8 Delay on the same path as Figure 8.6 for bidirectional packets. 196
8.9 (a) Packet delay for with (a) branch synchronization (b) regular synchronization. 197
8.10 Packet delay over the same path of the implementation (failed packets do not reach the eventual destination and are not counted in the average delay). 201
8.11 (a) Delay of non-urgent and urgent packets over a synchronized path. (b) Time line of transmissions of an urgent packet. 202
8.12 Packet pipelining: node 0 sends a new packet every $2t_i$. Transmissions end up being staggered. 203
8.13 Comparison of the lifetime of nodes running MiX-MAC with and without node synchronization. 204

8.14 (a) Successful path synchronization for two nodes sending data packets to a common destination. The packet rates are $^1/_{20}$ and $^1/_{10}$ $pkt.s^{-1}$. (b) The reduction in packet delay. 205

9.1 Representation of the system with input / output and its controller. . . . 209

9.2 (a) Evolution of $t_i(t)$ as the packet rate increases and then decreases when only packet loss is considered. (b) Packet loss in the same scenario. 213

9.3 (a) Nodes are in range of one another. (b) A receiver has several descendants. (c) A sender has several destinations (rare case). 218

9.4 Comparison of (a) the evolution of t_i and (b) the dropped packets for the controlled and non-controlled cases when the energy is evaluated. . 224

9.5 Comparison of (a) the evolution of t_i and (b) the dropped packets for the controlled and non-controlled cases when the energy is not evaluated as per Section 9.1.3.2. 225

9.6 Comparison of (a) the evolution of t_i, (b) the dropped packets and (c) the energy consumed for the controlled and non-controlled cases for two sources in a multi-hop network. 228

Chapter 1

Introduction

Wireless sensor network applications are becoming increasingly popular with the advent of affordable low power sensor platforms, or motes. The miniaturization of these platforms and the availability of numerous types of sensor nodes have allowed new deployments for increasingly complex applications. Various sensors may be attached to the motes, allowing the deployment of sensor networks in the fields of health care, room / building monitoring, and target tracking, among others. Because it is impractical and sometimes hazardous to replace batteries in the sensor nodes, the search for very low power hardware and protocols is still relevant today.

New protocols for sensor networks are frequently proposed, and many bring improvements in lifetime, quality of service to a user, or convenience to a programmer. Over the planned lifetime of a wireless sensor network, often measured in years, new protocols may replace older ones and provide increased performance or network lifespan. Thus it is important to allow improved protocols to replace existing ones, even in already deployed networks.

Over the years, cross-layer designs, which let two or more protocols from non-adjacent layers function in concert, have become very popular. Since these tend to sacrifice generality for performance improvements, the two goals of modularity, which provides flexibility in protocol updates, and specialization, which uses the specificities of a network to improve performance, are conflicting. With these considerations in mind, this thesis proposes a new sensor network architecture that standardizes cross-layer interactions, necessary to sustain further development of this field. In fact, modular cross-layering supports protocols that can save a great deal of energy to support service to the end user for longer periods of time.

Today, wireless deployments are ubiquitous and provide users with internet access or phone services almost everywhere. Information technology is now a new industry fueling human progress. With the right platform and architecture, sensor networks have a similar chance to spawn the next information revolution.

1.1 Overview of Wireless Sensor Networks

Wireless sensor networks are formed from small self-powered devices, consisting of one or more sensors, a microprocessor and one or several radio transceivers. The sensors include an analog-to-digital converter, and the microprocessor can be a microcontroller or a low power processor. The sensor node uses an energy source, usually with very limited capacity. It may not be possible to replenish the node batteries during the expected lifetime of the network.

The protocol stack is located in the software of the processor, except for the physical layer, which is typically implemented by the radio hardware and its controller. The other protocols, Medium Access Control (MAC), routing and transport, may be programmed using portable operating systems such as TinyOS [1] [2], Java [3], M∧NTIS OS [4], etc. To sustain the communication protocols, services are often required, for example services for finding the location of a node, its remaining energy, and link quality to neighbors. This thesis addresses how protocols and services are organized and how they can work together in a structured manner to serve the application and the end user with better quality of service (QoS) for extended periods of time.

Sensors in the network are able to *sense* the environment and report relevant data to a location capable of analyzing it. In the maturing technology of wireless sensor and actuator networks, some nodes may be fitted with devices that are able to *act* upon the environment they are in. Such networks may function in closed-loop form, without any intervention from humans. This is particularly needed in hazardous areas and remote locations.

Applications for wireless sensor networks are countless with existing deployments in the fields of environment monitoring including agriculture, industrial safety and manufacturing processes, health care, battlefield awareness, and many others.

For instance, sensor networks could be used to monitor patients from their homes. In addition to the evident health benefits for patients feeling more comfortable amidst the support of their families, remote monitoring could help cut costs by redirecting medical attention towards urgent care. Imagine that an oxygen sensor, a heart rate probe, a blood pressure gauge, an ECG sensor, etc. can be attached to a patient who is then discharged from the hospital. The combination of sensors can monitor his/her health continuously. In case of an emergency such as an abnormally high heart rate, medicine could be automatically administered to the patient through IV perfusion or drip. If the health of the patient is not observed to improve, the sensor network may summon an ambulance to the home of the patient. Inside the hospital, prognostic systems may warn doctors of the deteriorating condition of patients.

On a battlefield, soldiers may be equipped with the same health monitoring system, and their number could create a multi-hop network that relays critical information about the front line and the health of the troops. When the condition of a wounded soldier quickly deteriorates, the sensor network may relay information to the headquarters miles away. A medic could be sent out to the location of the person in need. Furthermore, the sensor network may help assess which critical areas are being weakened by the fast rate of casualties to a platoon.

As another application, consider new advances in the processing of materials. Manufacturers are engineering new alloys and resins for the space, aviation, and automotive industries. Aluminum foam [5] is being introduced into car manufacturing as a light component able to absorb high shock by collapsing millions of aluminum cells on themselves. Advanced metallurgy processes, such as the one for aluminum foam, require very precise procedures and concentrations. Sensors deposited at the bottom of a furnace may sense the exact composition of the compound in their direct location and prompt local actuators to introduce very fine amounts of the needed components.

These and many other applications all require a high degree of support for application QoS from the sensor network, while simultaneously requiring long lifetimes on the limited sensor batteries. Meeting these goals is a challenging problem, which is addressed in this thesis.

1.2 Research Motivation

Wireless sensor networks present many challenges that make research in this field both demanding and exciting. Obstacles include the following:

- Sensor nodes have very small energy supplies, which are either available from network deployment, or obtained by small amounts of energy scavenging.
- Wireless sensor networks are dynamic, either because nodes are mobile or because links between two neighbors may be broken or created. In harsh environments, nodes may also be disabled. The consequence is that communication between nodes is inherently unreliable.
- Individual sensor nodes have limited bandwidth, compute power, and memory, thus imposing specific constraints on programmers.
- Wireless sensor networks may comprise many sensor nodes; protocols need to scale to hundreds of nodes.
- The multiplicity of applications for wireless sensor networks has often led to a plethora of protocols whose information must be coordinated and processed.
- Wireless sensor networks are data-oriented, where individual nodes are only as important as their contribution to the application.

Following this last point, sensor networks support applications that require certain quality of service (QoS). Cross-layer improvements have been shown [6] [7] [8] to allow protocols to more closely meet the QoS requirements for extended periods of time.

Cross-layer designs may be best understood by explaining their opposite—layered schemes. The latter prevent communication between non-adjacent layers in the protocol stack and limit interactions between two adjacent layers to function calls and returns. Cross-layer protocols violate these principles and use information available at two or more levels in the stack to improve the network performance and / or lifetime.

At one extreme, the multiplication of cross-layer interactions within a protocol stack can lead to "spaghetti" designs, whereby the modification of one aspect in a protocol may have unforeseen consequences within many other protocols. Consequently, regulation of the exchange of information between protocols and the application is a key aspect of this thesis.

1.3 Research Contributions

This dissertation addresses the benefits and issues associated with cross-layer designs. The specific contributions include the following:

- The gains and limitations of cross-layer interactions are studied, and comparisons with layered interactions are performed to help draw conclusions as to which cross-layer interactions are beneficial.

- A new sensor network architecture called X-Lisa (Cross-Layer Information Sharing Architecture) is proposed. X-Lisa standardizes a specific type of cross-layer design, cross-layer information-sharing, while retaining a modular protocol stack. This architecture offers a new perspective on cross-layering by organizing the information shared between layers. It also lets each protocol focus on its core task, by organizing and providing key services to all protocols in the stack.

- While X-Lisa shows the advantage of adapting protocols based on current network conditions, protocols can similarly benefit from adaptation based on current application conditions and requirements. In particular, if protocols are proactively informed of the status of active queries in the network, they can adjust their behavior accordingly. Using this approach, the improvement in overall quality of service provided by the network can be significant. We augment X-Lisa with a Middleware Interpreter for managing application queries and performing event notification.

- With the availability of important network information at all layers in the stack, protocols may tune their internal parameters to improve network performance. An example of such protocol tuning is provided using a sensor network middleware that helps determine the nodes that are important to an application.

- Many improvements can be observed when using cross-layer information at the MAC layer. “Low-Power-Listening” (*LPL*) MAC protocols are investigated, and several techniques to adapt them to conditions on the network are proposed. Simple information about the packet size or the ratio of packets sent to packets received, can be used to improve the lifetime of a sensor node at practically no cost by appropriately selecting the MAC schedule of a protocol. Additionally, implicit synchronization of the wake-up schedules of nodes on a slowly changing path can be achieved without overhead with a subfamily of the *LPL* MAC protocols. With our third technique, dynamic adaptation of the duty cycle of these protocols can lead to substantially lower energy consumption and a higher packet delivery ratio. We simulated several *LPL* MAC protocols in Matlab, and we implemented them in TinyOS. Several interesting results arose from these experiments: we observed the differences seen between simulation and deployment results, inferred which assumptions were valid, and noted the importance of some design parameters.

1.4 Thesis Structure

Chapter 2 surveys the current state of research in relevant areas of wireless sensor network research and provides background on the state-of-the-art protocols to which our proposed schemes are compared. Chapter 3 evaluates the gains and limitations of cross-layer designs, and examines the source of the improvements. Chapter 4 presents X-Lisa, a new sensor network cross-layer architecture that enables both flexibility and protocol specialization. Following the observations of Chapter 3, X-Lisa provides a standardized model for cross-layer interactions between protocols, and it opens the door for new and improved protocols by sharing information with all the layers in the stack. This allows protocols to dynamically adapt to changing conditions in the network, providing better QoS or longer lifetime. Taking X-Lisa one step further, Chapter 5 presents a *Middleware Interpreter* that channels information between middleware and protocols to enable the protocols to constantly adapt to dynamic application requirements. The *Middleware Interpreter* proactively notifies protocols of query events, removing the burden of query monitoring and notification from middleware. In addition to conditions in the network, protocols can make decisions based on the application requirement. This allows middleware to convey the application needs to the protocols, which can tune *knobs* to meet the required quality of service. Chapter 6 studies one example, and shows how middleware can provide important information to cross-layer protocols. With all the information provided by X-Lisa and the Middleware Interpreter, protocol behavior can be improved greatly. One example is MiX-MAC, a novel *LPL* MAC protocol proposed in Chapter 7. MiX-MAC selects the MAC schedule best adapted to the conditions in the network. After adapting the MAC schedule of our protocols, we stagger node wake-up schedules to reduce packet delivery delay and energy consumption in Chapter 8. In Chapter 9, the duty cycle of the MAC protocol is dynamically adapted using principles of control theory. Together with MAC schedule adaptation and path synchronization, duty cycle control enables *LPL* MAC protocols to be utilized for networks with potentially high traffic loads. Chapter 10 concludes this dissertation.

Chapter 2

Related Work

This chapter provides background on the issues addressed by this thesis. We survey the literature related to cross-layer designs, protocols, services, and middleware.

2.1 Layered Protocol Stack

In this section, we provide background on various protocols used in a layered protocol stack, including node activation, routing and MAC.

2.1.1 Node Activation

Node deployments are usually non-uniform: node densities may vary over the region of interest, and not all the nodes may have the same sensing capabilities or compute resources. During the runtime, some nodes may have less remaining energy because of their location in the network (close to the stimuli or the base station, for instance). Redundancy between active nodes wastes precious energy of nodes that may be critical to QoS at a latter point in time. A node activation protocol turns off nodes that are not needed because the application goals are already met.

In [9], Ye et al. proposed using local redundancy to lengthen the network lifetime through a protocol called PEAS. The basic idea driving PEAS is that nodes located in densely covered regions can be turned off for long periods of time without significantly degrading the network coverage. All nodes go through a non-periodic wake-up round where they probe for active neighbors. If an active node overhears a *probe*, it sends a *probe reply* to the inquiring sensor. Only when the *probe* remains unanswered does the node activate—once active, it never goes back to sleep, unlike PECAS [10], an extension to PEAS by Gui et al. PEAS does not guarantee full network coverage as nodes only activate based on unanswered *probes*—which depend on the transmission range and not the sensing range.

2.1.2 Routing

Under the node activation protocol is the routing protocol whose role is to guide a packet from a source node to a destination, over one or more hops. Proactive routing protocols establish routes before a packet is even created and sent, while reactive protocols only do so as needed. In this section, we describe only a few of the many routing strategies that have been proposed.

Ye et al. proposed GRAB in [11], in which a cost metric field is established. Packet routing simply follows the path with steepest gradient that provides the minimum path cost. Because no consideration may be given to the quality of a link, unreliable paths may be used. GRAB routes packets along several parallel paths to increase robustness.

GFG [12] and GPSR [13] were seminal works in geographic routing. Packets are forwarded in a greedy manner towards the destination. Specifically, each node knows the location of its neighbors, and is able to elect a packet's next-hop as the closest node to the packet destination. If a node is the closest to the destination, but out of communication range, the right-hand rule applies: the packet received from a neighbor is transmitted to another node such that the angle between the three is positive.

In [14], Perkins et al. proposed AODV, a reactive routing protocol that establishes a route to a destination by flooding a route request "RREQ". Upon receiving a RREQ packet, nodes check their route tables for a known path to the destination. If one exists, they send a route reply to the source of the packet. AODV ensures that routes contain no loops and that the information between nodes is fresh.

2.1.3 Medium Access Control

MAC protocols help organize access to the common channel by neighbors in order to communicate. In this section, we only present MAC protocols that are related to the work addressed in this dissertation, specifically focusing on the family of “Low-Power-Listening” MAC protocols.

In [15], Langendoen surveys existing MAC protocols and proposes a taxonomy based on whether they utilize completely random access, slotted random access, or frames. Random access MAC protocols, of which Low-Power-Listening (*LPL*) and Preamble Sampling (*PS*) solutions are two examples, present the benefit of flexibility without the requirement for clock synchronization. *LPL* and *PS* differ in which existing MAC solution channel probing was added to: *LPL* is the result of a combination of channel probing with CSMA, and *PS* with ALOHA. Langendoen identifies the main downside of *LPL* and *PS* MAC protocols as often wasting energy in idle listening and collisions. Our work, described in Chapters 7-9 aims at correcting this drawback.

Aloha with Preamble Sampling (PS) was one of the first channel probing schemes proposed for wireless sensor networks [16]. In this approach, packets are sent with a preamble greater than or equal to the channel check interval t_i. Nodes periodically wake up and sense the medium. If the channel is busy, the probing node stays in receive mode until the data packet transmission is complete. Otherwise, the probing node goes back to sleep. El-Hoiydi developed an analytical model for Aloha with *PS* and studied its performance using four metrics: throughput, delay, power consumption, and lifetime. The transmit and receive powers assumed by El-Hoiydi in [16] led the author to recommend limiting the use of Aloha with PS.

Following Aloha with PS, El-Hoiydi et al. introduced WiseMAC [17], a MAC protocol that reduces the preamble length before sending a data packet by exchanging wake-up schedules between neighbors. However, WiseMAC as originally proposed (like B-MAC) cannot be implemented on 802.15.4 radios[1]. Moreover, piggybacking scheduling information to acknowledgment frames, as required by WiseMAC, supposes that hardware acknowledgments may not be used. Hardware ACK frames are considerably faster than software acknowledgments (by a factor of two to five depending on packet size and code optimization) and allow the radio to return to sleep much earlier, resulting in significant energy savings. For these reasons, and because it is classified under the *PS* family of MAC protocols, it was not included in most of our study

---

[1] An updated version of WiseMAC was proposed for 802.15.4 compliant radios, where a repetition of frames, similar to SpeckMAC’s schedule, replaced a long preamble.

in Chapter 7. We show that explicit scheduling between nodes is unnecessary because it can be achieved implicitly with X-MAC, C-MAC, or MX-MAC.

B-MAC [18] with Low-Power-Listening (LPL) was the first MAC protocol to introduce channel probing schedules for recent radios. Polastre et al. provide a model for *LPL* with strong consideration for the target radio. The authors thoroughly compare B-MAC to S-MAC [19] and T-MAC [20]. To curb limitations imposed on the receiving node to stay awake for the time of the preamble, Polastre et al. propose sending packets with half-sized preambles. Post-B-MAC protocols include X-MAC [21] and SpeckMAC-D [22]. Both protocols are of the channel probing family and tried to improve the low power listening scheme presented by B-MAC. They are discussed in detail in Chapter 7.

Although more recent, C-MAC [23] uses the same schedule as X-MAC and is therefore included in our work under the same principles that govern X-MAC.

In [22], Wong and Arvind also propose SpeckMAC-B, which is compared, along with SpeckMAC-D, to B-MAC. The SpeckMAC protocol family is intended for miniature motes called specks and will be detailed in Chapter 7. SpeckMAC-B stands for *Back off* and replaces the long preamble with a sequence of wake up packets containing the destination target and the time when the data packet will be sent. This allows receiving nodes to sleep for the remainder of t_i and activate just in time for data reception. However, this scheduling assumes fine time synchronization between nodes, which we do not presuppose in our work. Wong and Arvind develop a model for SpeckMAC protocols and study their impact on the ProSpeckz platform (a larger speck of size one square inch), while comparing them to B-MAC.

Finally, Lu et al. proposed DMAC [24], a MAC protocol whose goal, much like ours in Chapter 8, is to stagger wake-up schedules over paths of a data-gathering tree. DMAC defines receive and transmit slots for unicast packet exchange at every node. In order to achieve synchronization, the slots are staggered along paths through the explicit exchange of schedules among neighbors. Because a node must know the Rx / Tx slots of its neighbors, DMAC requires local time synchronization. Additionally, broadcast packets from the data sink to the leaf nodes are only supported in specific slots, which may cause increased latency and energy waste when these slots are not used. Our transmit / receive schedule synchronization approach conserves synchronization for these centrifugal flows. The improvements obtained by DMAC are significant and convincing, and we faced many of the same hurdles as Lu et al. However, our scheme achieves similar results *without* the overhead and limitations supposed by DMAC and comes at specifically no additional cost when a subset of the *LPL* MAC protocols are used within the scope of applications chosen for D-MAC.

Much like DMAC, Keshavarzian et al. propose to stagger wake-up schedules for random access protocols [25]. Several cases, depending on the direction (forward or backward) of the traffic, are considered. In the forward case, all nodes wake up at the same time every $T\ s$ causing large delivery delays of hT, where h is the number of hops. The "shifted even and odd" pattern alleviates this problem somewhat: all nodes shift their wake-up schedule by ${}^{T}/_{2}$ with respect to their previous hop. At worst, the delay is $(h+1)^{T}/_{2}$. The ladder pattern, much like DMAC, forces nodes to wake up only a few milliseconds after their previous hop. Multi-parent ladders accounts for the case when networks have multiple branches: the inter-wake-up time T is divided to accommodate various branches. Whether compared to this work or to DMAC, our scheme achieves similar results *without* the overhead and limitations supposed here and comes at specifically no additional cost when the already popular X-MAC, C-MAC or MX-MAC *LPL* protocols are used within the scope of applications chosen by Lu et al. and Keshavarzian et al.

2.2 Cross-layer Protocol Stack

In [26], Srivastava et al. provide a definition of cross-layer designs and a survey of existing cross-layer models. The authors define cross-layer interactions as back-and-forth information flows, merging of adjacent layers, design coupling without a common interface, and vertical calibration across layers. They also list implementations for cross-layer interactions: explicit interfaces between different layers, shared database, and heap organization. Chapter 4 shows that our work falls into the second category by defining a set of data structures and services, and it offers an information propagation paradigm as well as lower level abstractions.

Figure 2.1 shows an example of a layered structure: communication between protocols are limited to adjacent layers and through function calls and returns. Cross-layer

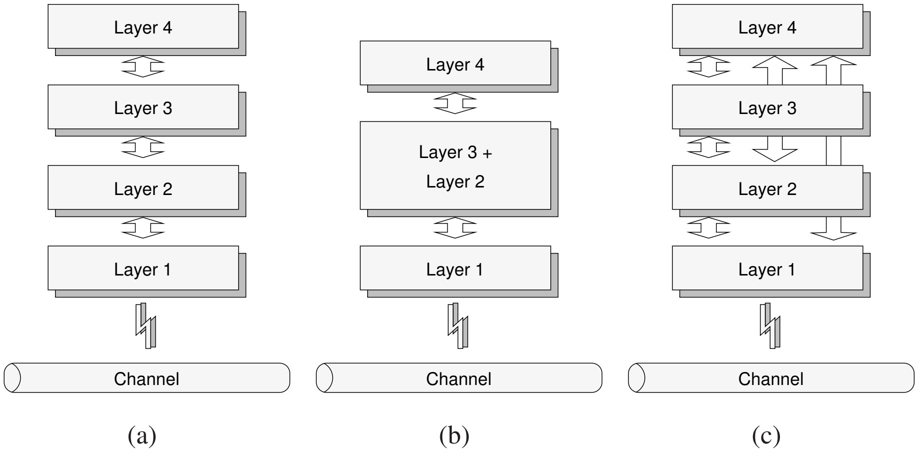


Figure 2.1: (a) Layered structure. (b) Cross-layer structure with fused layers. (c) Cross-layer structure with information exchange between layers.

structures violate this principle.

2.2.1 Cross-Layer Protocol Limitations

There are several reasons that motivate violating layered architectures: unique hurdles present only in wireless sensor networks, opportunities created by the spatial concurrency of the medium, and specific goals of the various applications and end users. However, the multiplication of interfaces and direct links between layers has many drawbacks, such as:

- Architecture violations render the protocols hard to maintain and complicate *in situ* updates.
- Some violations can have unforeseen effects on the performance of the network if they are not carefully studied, as was argued in [27].
- If too many architecture violations are introduced in the design, the architecture no longer represents the actual system and its significance is weakened.

Work by Kawadia et al. [27] offers a concrete comparative look at cross-layer and layered designs. This work makes the case that unintentional cross-layer interactions can have detrimental consequences on system performance. Creating a cross-layer architecture requires caution because of the numerous interactions of the layers (creating a spaghetti design).

Some cross-layer protocols have nonetheless provided improvements over layered designs.

2.2.2 Cross-Layer Protocols

In Chapter 3, we compare PEAS to DAPR [6] (Distributed Activation with Predetermined Routes) which is a joint routing and node activation protocol proposed by Perillo et al. In [6], the target application is area coverage. DAPR assigns an application cost that quantifies the importance of a node to the coverage task: the higher the cost, the more important a node is to the coverage of a sparsely monitored area. Before packets can be routed, the data sink floods a *query*, which sets up routes with smallest cumulative costs. Packets are then routed along paths of smallest cost, while nodes whose coverage is not required deactivate, the ones with higher cost having the first chance to turn off. Through cross-layer improvements, the network is able to serve the application longer by utilizing nodes in areas that are more densely monitored. Additional details of DAPR are provided in Chapter 3. DAPR is an example of layer fusion, as illustrated by Figure 2.1(b): node activation decisions are based on metrics used by the routing protocol. The two functions are also coordinated in a cross-layer fashion.

In [28], the MAC layer is a time-division based scheme where slots are further divided in sections, one of which contains control information that enables neighboring nodes to synchronize appropriately. The routing protocol (ESR) builds a connected set of active nodes, each sensor making a decision on its own role based on local information exchanged in the control section (TC) in time slots of the MAC layer. In addition, the routing protocol is notified of a link break by the MAC protocol within one TDMA frame. A third layer benefits from MAC protocol information: the node activation protocol also receives schedules of the node's neighbors, which are obtained in the TC subframe. Cross-layer interactions can be seen between the MAC and routing protocols, and the MAC and node activation protocol, in the fashion described by Figure 2.1(c).

In [29], Lin et al. study the effects of a distributed and imperfect cross-layer rate control scheme, and compare its results to a layered architecture. The layered technique sets the rate control using prior knowledge of the capacity in a region and does not use the dynamics of other layers. The cross-layer architecture sets both the data rates and the resource allocations at the physical and MAC layers. The cross-layer approach greatly outperforms the layered one.

2.3 Adaptability in Wireless Sensor Networks

Much work has been dedicated to the task of adapting MAC protocols to conditions in the local neighborhood of a node [20] [30] [31]. The authors in [30] aim to adapt the data rate for packet transmissions based on channel quality predictions in high-rate WPANs. The intended destinations of a prospective sender are notified through *rate-adaptive ACK* frames. In [20], van Dam and Langendoen propose T-MAC to improve S-MAC by a novel adaptive active/sleep duty cycle. T-MAC sends packets in bursts during active periods, and if no activity is detected during a small window of time, nodes return to sleep. As a consequence, nodes have a shorter duty cycle under T-MAC. In [31], Pham and Jha introduce MS-MAC, an S-MAC based protocol that adapts S-MAC's listening, sleeping and synchronization cycles to anticipated node movements. Node displacement is calculated from changes in signal strength; in case of rapid movements, a sending node hastens packet transmissions before a connection is lost.

Jurdak et al. [32] introduced the idea of adaptive duty cycles in *LPL* protocols. Because a protocol designer must account for busy regions of the network, a fixed t_i value would have to be set conservatively. Consequently, many parts of the network would waste energy by running at an unnecessarily high duty cycle. Adaptive Low-Power-Listening, or ALPL, allows areas of the network to run at a lower duty cycle. After forming their routing tree, each individual node can evaluate the number of packets they will transmit per second based on the expected number of packets they will originate and that of their descendant nodes. These values are periodically announced by the nodes. The further away from the data sink, the fewer children a node has, and consequently, the smaller the packet rate it is expected to carry. Its duty cycle can thus be lowered to a smaller value than that of nodes closer to the data sink. Contrary to ALPL, our approach does not use a heuristic and adapts the duty cycle using control theory techniques to meet the target rate of packets. Chapter 9 studies the consequences of adaptive t_i values over the network.

The idea of using control theory in sensor networks is not a new one, especially because wireless sensor and actuator networks require such solutions. In our unique approach we optimize the duty cycle for both energy use and packet transmissions, which cannot be easily modeled. Examples of existing methods that use results of control theory to adapt parameters in a Wireless Sensor Network (hereafter, *WSN*) include [33] and [34].

In [33], Vigorito et al. use control theory to adapt the duty cycle of nodes capable of harvesting energy. Maintaining a sufficient power supply level is a non-trivial problem because of changing environmental patterns such as the weather. The authors introduce a model-free approach to adapt the duty cycle in dynamic conditions. Although they set out to control only one parameter in the system (the energy supply level), which constitutes a marked difference from our goals, much of their underlying theoretical foundations are similar to those in the first part of our work described in Chapter 7.

In [34], Le et al. propose to optimize channel assignment to increase the throughput in multi-channel WSNs using a control theory approach. The throughput on individual channels can be easily modeled with the nodes' individual load, which includes that of its descendant nodes. When the total load M_i on channel i is above its optimal value M_r (one that guarantees little contention for instance), nodes transmitting on this channel may change to another channel $j \neq i$ with a probability proportional to the difference (error) between M_r and M_i. Le et al. also account for delay, which can cause overshooting and undershooting—instability of the system response.

2.4 New Architectures

This section provides an overview of related work on cross-layer protocol architectures and briefly outlines some of the architectures that will be surveyed in more detail in this work. We also include protocols that have such a profound design impact that they inevitably introduce a new architecture.

Whitehouse et al. introduced Hood [35], a neighborhood abstraction for WSNs that allows nodes to identify neighbors with variables of interest. Nodes define *attributes* that may be shared with neighbors. Upon receiving attributes, each node evaluates whether these are valuable enough to be recorded in a neighbor list. Because it is not an architecture, it is not part of this study. However, the solution retained to manage its neighbor table is similar to ours.

In [36], Wang et al. survey existing cross-layer signaling methods, which most closely corresponds to the explicit interfaces architecture mentioned above. Additionally, the authors propose CLASS (Cross-LAyer Signalling Shortcuts), an architecture that allows propagation of cross-layer messages through out-of-band signaling. Because Wang et al.'s work already surveyed the state of this type of architecture extensively, the focus of our work does not include these architectures.

Conti et al. proposed MobileMan [37], an architecture for MANETs. MobileMan extends cross-layering to all network functions through data sharing of local network status. The goals of MobileMan include modularity, standardization, support for inter-compatibility and maintenance. The philosophy behind MobileMan is to adapt the Internet protocol architecture to MANETs while allowing cross-layer designs.

Sadler et al. [38] propose a shared *platform* among all layers of a protocol stack for cross-layer optimizations. The authors recommend using a table of interchangeable nodes (capable of handling the same application request) that lists equivalent nodes to be used by the routing protocol when a link breaks. The criteria used are connection oriented since the work focuses on providing reliable links in MANETs. While the *platform* certainly introduced architecture choices made by Sadler et al., the paper focuses on protocol issues rather than a universal architecture. The principles guiding the shared *platform* are, however, well represented in the surveyed architectures and are the object of further study in our work.

In [39], Winter et al. propose CrossTalk, a cross-layer architecture for mobile ad-hoc networks that aims to provide a global view of the network to individual nodes. CrossTalk disseminates information over a full path, accumulating information about sensor nodes along the way. Information is provided to all protocols in the stack. Even though CrossTalk provides only pieces of information along data paths, the network view is above 95% accurate.

In [40], Culler et al. propose SNA, a new architecture whose goals are to provide data link abstraction and a modular network layer. Culler et al. broke protocols down into fundamental functions that allow code-sharing and reuse. SNA also relies on a sensor network protocol (SP) [41] and creates an abstraction level above the Data Link layer to compensate for the multiplicity of platforms available today. SP uses information repositories: a neighbor table and a message pool. Being a protocol, it also makes decisions such as packet reordering and forwarding.

Dunkels et al. propose Chameleon in [42], an architecture that allows the application and protocols above the transport layer to benefit from abstracted underlying communication mechanisms. The architecture uses a small yet expressive packet header to convey all communication patterns common to WSNs. This solution falls into the vertical calibration category as defined by [26].

XLM [43] takes the reverse approach from the previous architectures by fusing all communication layers into one. This helps ensure that all functions in the stack are performed in a coordinated and sensible way. However, XLM has to assume that only CSMA-based MAC protocols will be used. The natural presumption of this idea is that this family of MAC protocols is the most energy efficient for a given QoS. Even though XLM was not proposed as an architecture, but merely as a highly integrated protocol, the proposed changes are so pervasive that they incur fundamental architecture redesign. XLM provides an interesting counter-point to information sharing among all layers, even though the main goal of Akyildiz et al. was to maximize network lifetime. The fused layers necessarily see the same information, much like layers of the other architectures that dispose of the same data.

2.5 Services

Much work has been dedicated to developing and refining important services to improve network protocol performance, such as time synchronization [44] [45], localization [46], ID assignment [47] and entity assignment [48].

In [44], Römer proposes a time synchronization scheme that uses clock drift and the exchange of two packets. The idea is *not* to synchronize local computer clocks, but to measure time differences between packet transmissions. Figure 2.2 illustrates the principles behind the algorithm.

RBS [45] allows for more precise time synchronization by stamping the time at

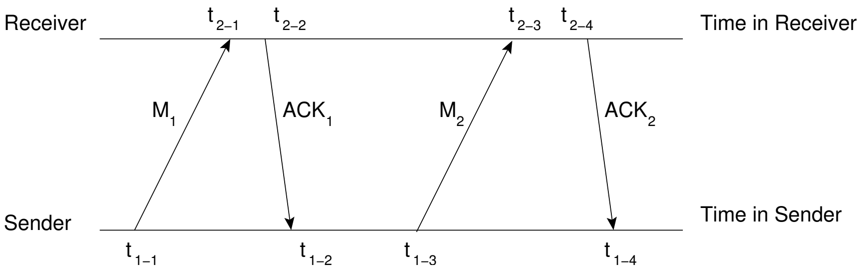


Figure 2.2: Time diagram of the message delay estimation. The sender may measure the delay as $(t_{1-2} - t_{1-1}) - (t_{2-2} - t_{2-1})$ or $(t_{1-4} - t_{1-3}) - (t_{2-4} - t_{2-3})$. The receiver can estimate the packet delay as $(t_{2-3} - t_{2-2}) - (t_{1-3} - t_{1-2})$.

which a packet is received at the lowest level possible. Since most non-deterministic delays in transmitting a packet lie between the layer originating a packet and the radio (next-hop determination at the routing protocol, queue and MAC delays, etc.), RBS removes much of the uncertainties associated in time synchronization. The algorithm starts when a transmitter sends m broadcasts. Its neighbors then exchange the times at which they received the m broadcasts with one another to synchronize their clocks.

In [47], Ould-Ahmed-Vall et al. propose a distributed algorithm that assigns globally unique IDs coded over the smallest number of bytes achievable to limit header sizes. A tree structure is first organized using temporary IDs; parent nodes then assign IDs to their children. In [48], Blum et al. propose middleware that forms and maintains a unique *entity* around an *event*, allowing abstraction of the transport layer between different entities.

In [49], Lenders et al. propose a publication service for data retrieval in the network. To achieve this, two types of identifiers are introduced: the first, a *service type identifier* (STID), and the other, a *unique identifier* (UID). Nodes providing the same types of service on the network necessarily have the same STID. UIDs are only issued before requesting a service that requires unique identification. Routing was inspired by GRAB [11] and based either on STID fields, or UID fields depending on the needs of individual nodes.

In [50], Stoica et al. propose $i3$ (Internet Indirection Infrastructure), a communication service that decouples the act of sending and receiving. When a sensor node has data that it wants to share, it sends its node ID and the data to a logical identifier in the local network. The identifier simply matches the posted data to *triggers*, which are sent by a receiver interested in specific information. This *rendez-vous* based communication abstraction provides support for mobility, anycast and multicast.

2.6 Middleware for Managing Cross-Layer Protocols

In [51], K. Römer et al. offer a look at challenges in designing middleware for sensor networks. They identify the need for data-centric communications, adaptive fidelity functions, and QoS knowledge. They also define middleware tasks as assigning high-level and complex sensing formulation and adaptation to the network. Our research mostly follows the precepts delineated by K. Römer et al.

Wang et al. provide a survey of middleware for WSNs in [52]. They analyze middleware projects in terms of *programming abstractions* (how the programmer views the system), *system services* (support for application deployment, execution, and network management), *runtime support* (management and redistribution of resources when the node cannot provide them), and *QoS mechanism* (interaction between the application and the network infrastructure, usually in the form of cross-layer components).

Among existing middleware for QoS mechanism, Wang et al. cite MiLAN [53], whose role is to map application requirements to the nodes' sensing capabilities so as to provide the exact QoS required by the application. MiLAN [53] is a *proactive* middleware that controls the sensors to adapt the network to the application needs as they vary over time, specifying which sensors should send data, route packets, etc. MiLAN allows the network protocols and the application to communicate thanks to various graphs. The *state-based variable requirements* graph specifies what variables are needed under different states of the monitored environment. The *sensor QoS* graph communicates the abilities of each sensor node in providing QoS. Heinzelman et al. also propose a taxonomy of middleware in [53] and classify MiLAN as adaptive and proactive middleware for dynamic environments.

Other middleware techniques manage resources proactively: *e.g.,* DSware [54], MagnetOS [55], AutoSeC [56], etc. DSware does not require or allude to vertical interactions with the protocol stack. MagnetOS, although a Java-based operating system, carries middleware tasks to redistribute application components within the network.

Even if direct interaction with the protocol stack is not supported, this redistribution has a direct impact on network protocols due to the shifting of communication endpoints. Conversely, AutoSeC specifically interacts with the protocol stack. It manages network resources by collecting data from various protocols and services. However, this relation is not bidirectional since middleware decisions are not explicitly fed back to the protocol stack.

Most closely related to our work with middleware, described in Chapter 5, is Impala [57]. Liu et al. proposed a middleware system, articulated around a new architecture that specifically focuses on middleware. Their design goals mirror many of our own (modularity, ease of updates, energy efficiency, etc.). Much of Liu et al.'s work focuses on replacing pieces of code on the fly. Impala also lets the application dynamically adapt to improve QoS and energy efficiency through a dedicated interface. Packet-, data- and device-related events are provided to the application through this interface, which filters and dispatches these events to the relevant application. The model proposed in Chapters 4 and 5 differs in that a similar interface extends to all layers of the stack, and that the flow of information is bidirectional. Moreover, data filtering is done by checking whether an application query has fired before it is dispatched (a new sensor value will not necessarily create a new event if it does not match the conditions of a query).

Y. Yu et al. present a cluster-based middleware in [58] and provide a level of abstraction that separates application semantics from the layered protocol architecture. They introduce a cluster forming layer (within the middleware virtual machine) responsible for building clusters around a phenomenon in a distributed manner. They define knobs or mechanisms designed to change the working mode of a sensor. These knobs are similar in spirit to the tunable parameters presented in our work.

In [59], Cornea et al. study middleware optimizations for cross-layer architectures dedicated to interactive mobile video streaming. The middleware identifies the viewing device and forwards this information to the nodes, who then adapt their sleep and active periods, bit rate, frame rate, and video resolution to the needs of the viewing device. Low-level optimizations are thus integrated with a middleware to enhance the user experience.

Chapter 3

Gains and Limitations of Cross-Layer Designs

3.1 Introduction

Many researchers have conjectured that wireless sensor networks can particularly benefit from cross-layer architectures due to the inherent resource constraints of such networks and their application-specific quality of service metrics. Although more integrated designs lose generality and increase complexity, they can be fine-tuned to better serve a specific application.

One such cross-layer protocol for wireless sensor networks is DAPR (Distributed Activation with Predetermined Routes) [6], which integrates sensor activation and routing functions. DAPR is based on the intuition that sensors whose data are less important to the application should be likely candidates to serve as routers. DAPR jointly selects which sensors should be active to cover the area with minimal redundancy and the routes packets should take from these active sensors to the base station. Alternatively, these functions of sensor activation and routing can be performed in a layered architecture as well. For example, PEAS [9] is a sensor management protocol that exploits redundancy in densely covered areas to allow sensors to sleep and thus guarantees a longer network lifetime. PEAS can be used with any standard routing protocol, such as AODV.

The goal of the work described in this chapter is to study the respective advantages and drawbacks of cross-layer and layered architectures, and hence to verify or rebut the conjecture that cross-layer designs are well-suited to wireless sensor networks, through a comparison of cross-layer and layered protocols. To this end, we use NS-2.28 [60] simulations of cross-layer DAPR and PEAS+AODV with various network, sensor, and phenomena scenarios. The application goal is to provide geographic coverage of the network over time, as this was the original design goal for both DAPR and PEAS. However, this metric can be generalized by redefining ‘application coverage’ for heterogeneous sensors—for example, by considering the ‘coverage’ of a person’s vital statistics.

The results of our study lead us to argue for the implementation of certain types of cross-layer optimizations, which can be facilitated through a new sensor network architecture that would take advantage of tunable parameters present in every protocol.

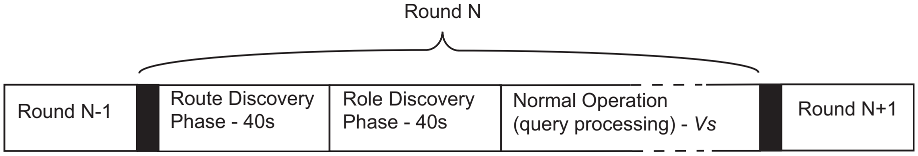
3.2 DAPR: A Cross-Layer Protocol

DAPR quantifies the importance of each sensor to the application to avoid using key nodes as routers. Each sensor is assigned an ‘application cost’ that reflects the criticality of the node’s data. This application cost is shared among the routing and node activation routines.

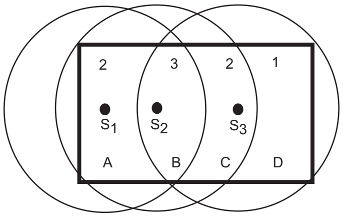
In DAPR, the base station, which also serves as the data sink, sends periodic queries that trigger a route discovery phase and a role discovery phase. These two phases and the query reply period form a round during which routes and sensor activations are typically fixed. Figure 3.1(a) shows the timing of DAPR rounds.

3.2.1 Route Discovery Phase: Selecting Minimum Application Cost Paths

In DAPR, the application cost measures the comprehensive coverage in the monitoring area of a sensor. This cost is defined as the inverse of the coverage provided by all sensors capable of sensing each sub-region (assessed in terms of total energy) of the sensor’s coverage area, normalized by the area of each sub-region. Formally, applica-


Round N
Round N-1
Route Discovery Phase - 40s
Role Discovery Phase - 40s
Normal Operation (query processing) - Vs
Round N+1

(a) A DAPR round.



(b) Sensor network coverage of a rectangle region.

Figure 3.1: (a) DAPR rounds consist of three phases. (b) A sensor checks whether its coverage is overlapping with that of others during the role discovery phase. Sensor 2 is not needed to cover the rectangular area.

tion cost is defined as:

$$C_{ac}(S_i) = \int\limits_{R(S_i)} \frac{dx}{E(x)} = \int\limits_{R(S_i)} \frac{dx}{\sum\limits_{S_j : x \in R(S_j)} E_{res}(S_j)} \tag{3.1}$$

where $C_{ac}(S_i)$ is the application cost for sensor S_i, $R(S_i)$ is the coverage area of sensor S_i, $E(x)$ is the total energy of all sensors capable of monitoring point x, and $E_{res}(S_j)$ is the residual energy of sensor S_j. Figure 3.1(b) provides an example to illustrate the computation of application cost. In this example, each sensor has a residual energy of 1, *i.e.*, $E_{res}(S_i) = 1$. S_1 has a cost of $C_{ac}(S_1) = \frac{A}{2} + \frac{B}{3}$, and S_2 and S_3 have costs of $C_{ac}(S_2) = \frac{A}{2} + \frac{B}{3} + \frac{C}{2}$ and $C_{ac}(S_3) = \frac{B}{3} + \frac{C}{2} + D$, respectively, where A, B, C, and D designate the areas of the same name.

The base station periodically floods a query indicating its interest for sensor reports and helps disseminate the nodes' information. Sensors add their individual cost to the current path cost to the base station, setting up routes as the queries flood through the network. A node forwards a query after a delay proportional to the cumulative application cost. This guarantees that a node always receives a first query through the path of lowest cost. Later in the round, when sending a packet, each node selects its neighbor with the lowest cumulative cost to the base station as the next hop. Consequently, DAPR avoids routing packets through nodes whose coverage is critical.

3.2.2 Role Discovery Phase: Providing Maximum Coverage

When parts of the region of interest are densely covered, there may be many nodes whose sensing area is fully redundant with the coverage provided by other active sensors. These sensors can be turned off without degrading the global coverage, and they can be activated later in time when their neighbors die and the network coverage becomes compromised. After the route discovery phase, nodes assess the necessity of their own coverage after a delay inversely proportional to their cumulative cost: nodes with a high cost consider turning off first. If a node determines that its data is redundant with that of other active nodes, it beacons to its neighbors to indicate that it is deactivating. In the case of a monitoring application, a node is not needed if its sensing area is overlapping with already active nodes.

3.2.3 Integration with the MAC and Physical Layers

DAPR was originally designed to work with a TDMA-based MAC (referred to as RT-dma) that enables nodes to sleep during periods of inactivity. The cross-layer design of DAPR can be extended to the MAC layer, as seen when a node beacons to signal it will be turning off during the round. A node checks whether its sensing area is redundant at the MAC layer, rather than at the routing or sensor activation layers, to avoid problems incurred by the MAC queuing delays. This ensures that a node will not improperly turn off because it did not receive all beacons due to MAC queuing delays. To illustrate the further advantages of integrating the MAC layer with the upper layers, we added support for DAPR to work on top of IEEE 802.11 in NS-2.28. Another level of integration appears at the physical layer where transmission power control is used. This physical layer uses only the power necessary to reach the destination node, ensuring precious energy savings.

3.3 Simulation Scenarios and Results

While most researchers believe that cross-layer architectures can enable networks to operate better for specific applications, it is not known exactly how much improvement can be obtained using a cross-layer architecture, under what conditions these benefits can be obtained, what the trade-offs are, or if a smart (layered) design can provide sufficient gains. The goal of this section is to provide a thorough comparison of one cross-layer architecture (DAPR) and one layered architecture (PEAS+AODV) to try to answer these questions through evaluation of a data point in the architecture space.

3.3.1 Methodology: the Discrete and Continuous Cases

PEAS and DAPR were designed with distinct goals for the sensor network application. While PEAS was proposed with the aim of tracking one or several targets in a region, the main objective of DAPR was to deliver periodic data reports from all nodes monitoring the network. We refer to the first goal as *discrete monitoring* and the second goal as *continuous monitoring*.

3.3.1.1 Discrete Monitoring of the Region

For simulations utilizing the discrete monitoring mode, only one (mobile) target is introduced in the network. While several nodes might sense the presence of the target, only one report is generated and sent to the base station. In the case of PEAS, the node closest to the stimulus is elected to send the report in keeping with Ye et al.'s original scheme, whereas DAPR selects the sensor with the smallest cumulative cost to the data sink. New sensor reports are obtained every $10\ s$. The coverage of the network is defined as the collective percentage of the region monitored by active sensors.

3.3.1.2 Continuous Monitoring of the Region

For simulations utilizing the continuous monitoring mode, all active nodes send one report every $10\ s$ to the data sink. This application requires that coverage be defined as the cumulative areas covered by nodes whose data packets are successfully received at the base station. As a consequence, nodes that are active but belong to a partition that excludes the data sink are not counted in the calculation of coverage. This coverage only differs from the *discrete monitoring* one when the node density is very low—typically true at the very end of the network life.

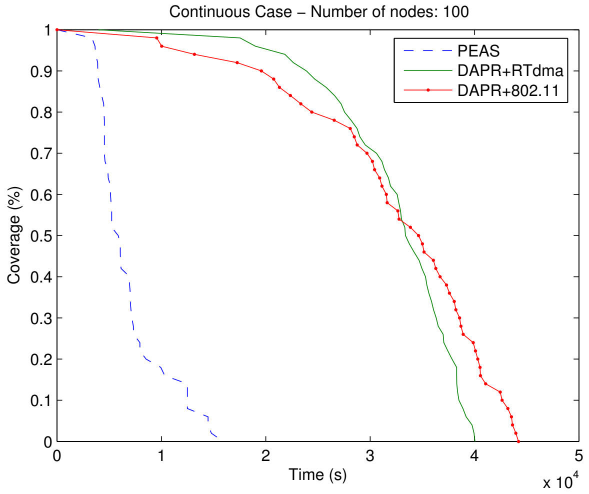
3.3.1.3 Network and Protocol Parameters

We ran simulations on circular networks of radius $100\ m$ for a coverage-based application. Nodes were placed according to a uniform random distribution. The maximum transmission range was set at $100\ m$. With these parameters, path lengths of 4 or 5 hops were typically observed with DAPR. Our results compare DAPR with RTdma, DAPR with IEEE 802.11, and PEAS with AODV and IEEE 802.11. We use the energy model from [61] where the transmission and reception powers are $6\ mW$ and $5\ mW$ at $10\ m$. The DAPR route and role discovery phase was set to $80\ s$, and the total round length was set to $500\ s$. A short round length (*e.g.*, $500\ s$) was found to yield a better network lifetime at 100% coverage at the expense of reducing network lifetime for lower coverage. In essence, the longer the query interval, the more similar DAPR behaves to a layered architecture, and since this work compares layered and cross-layer designs, we selected a short round length. While DAPR currently features a fixed round length, an application might need a changing query interval based on application or environmental states; we address this need and the means to control the round length through a middleware system in Chapter 6.

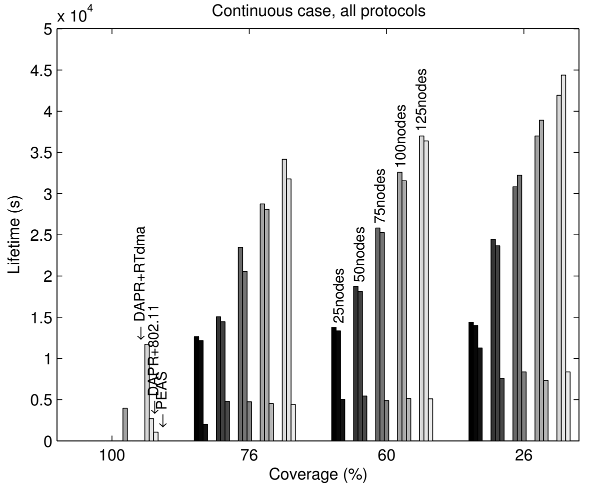
3.3.2 Continuous Monitoring of a Region

3.3.2.1 MAC Layer Improvements

For the sake of exposing cross-layer improvements at the MAC layer, we ran simulations with a modified application cost. In this case, we use a very common energy cost, $\frac{1}{\epsilon_{rem}}$, *i.e.*, the inverse of the sensor's remaining energy. Figure 3.2 shows the coverage percentage as a function of the last time it was reached. The energy savings from RTdma enable the fully integrated version of DAPR (DAPR+RTdma) to outperform DAPR+802.11, as seen in Figure 3.2(a). Nodes with a TDMA MAC layer can conserve more energy by sleeping during periods of inactivity; also, they do not perform an RTS / CTS / ACK handshake when they send a unicast packet. As the number of nodes grows, full coverage is provided for a longer time with DAPR+RTdma (Figure 3.2(b)).



(a)



(b)

Figure 3.2: Continuous monitoring: network coverage over time for modified DAPR ($C_{ac}(S_i) \propto \frac{1}{\epsilon_{rem}}$) and PEAS, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 100 nodes. DAPR outperforms PEAS, and the contention incurred by 802.11 causes nodes to wrongly decide to sleep in the first moments of the simulation. DAPR+RTdma, as a fully cross-layer design, prevents deactivation mistakes.

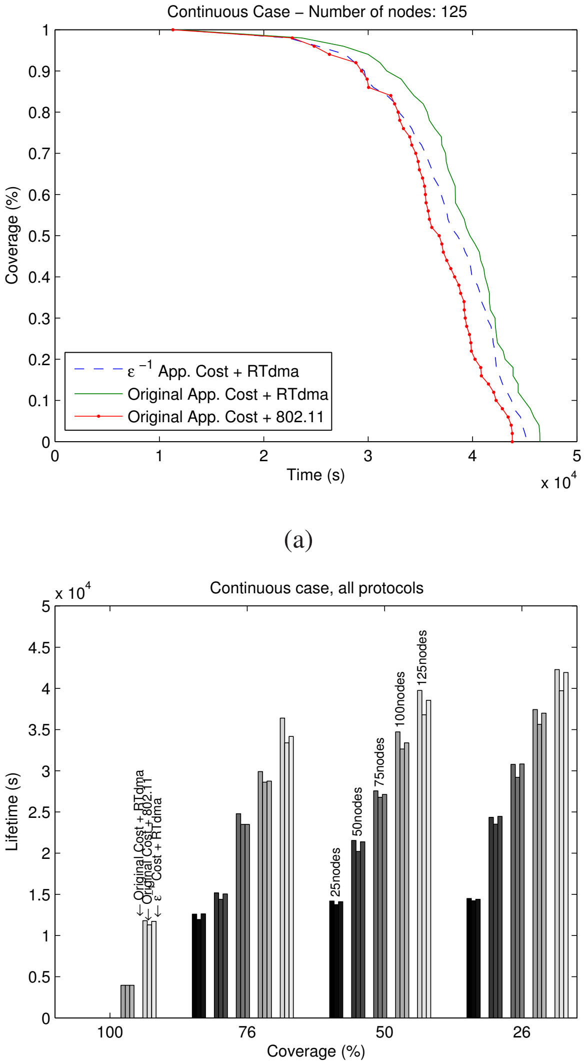
Additionally, we found that when many nodes have roughly the same application cost, as is true at the beginning of each simulation, IEEE 802.11's contention incurs delays in the delivery of beacons. This results in less than optimal coverage in the first minutes of the simulation but is compensated by better coverage later in time as nodes die and contention is eased. Fully cross-layer designs avoid this problem.

On the other hand, PEAS' performance is more modest: the coverage, while reaching 100% for the first $2,500\ s$, decreases rapidly around $5,000\ s$. The constant packet sources add significant contention to the network, especially in dense networks; probes and / or probe replies are either lost or not received in time, causing more sensors than necessary to turn on. After a while, all nodes are activated and cannot go back to sleep, as PEAS has no provision to turn off a node. This results in a premature loss of coverage. This characteristic toss up between the number of nodes and the coverage is shown in Figures 3.2(b) where larger numbers of nodes do not necessarily translate into longer lifetimes for PEAS.

3.3.2.2 Gains Are not Always Obtained from Cross-Layering

With DAPR's original application cost ($C_{ac} \propto \frac{1}{\Sigma\epsilon_{rem}}$) as defined in Section 3.2, the network lifetime improves by 3% to 15% for levels of high-range to mid and low-range coverage (shown in Figure 3.3). It can be inferred from Figures 3.3 and 3.4 that an even greater improvement is obtained with DAPR+802.11 because only a few nodes have the same application cost—which depends on the number of direct neighbors. As a result, nodes do not evaluate opting out at exactly the same time, relaxing the MAC incurred delays.

While cross-layer gains are possible and substantial as seen in the previous subsection, much of these gains can be achieved simply by selecting a carefully chosen application cost. In the example provided above, cross-layering can help alleviate some of the protocol stack's shortcomings—such as MAC queuing delays. Figure 3.4 shows that, on the other hand, DAPR with its original application cost and IEEE 802.11 does not suffer from the queuing delays that were observed in Figure 3.2. The only observable difference between DAPR+802.11 and DAPR+RTdma stems from the MAC overhead.



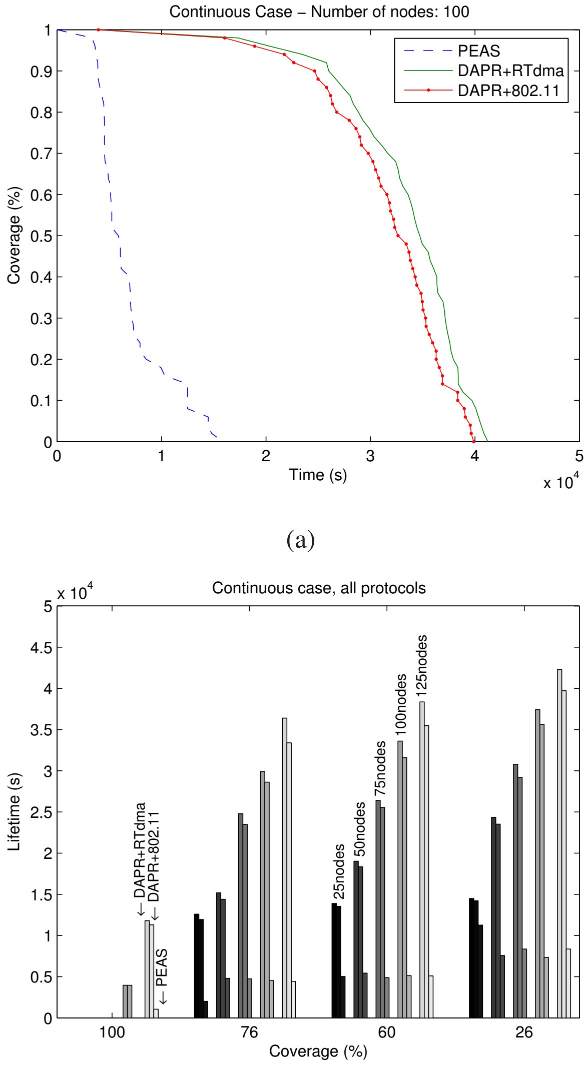
(a)

(b)

Figure 3.3: Continuous monitoring: network coverage over time for DAPR with its original ($C_{ac}(S_i) \propto \frac{1}{\Sigma \epsilon_{rem}}$) and modified ($C_{ac}(S_i) \propto \frac{1}{\epsilon_{rem}}$) costs, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 100 nodes. With RTdma, the original cost allows for gains of 3% to 15% over the energy cost.

3.3.3 Discrete Monitoring of a Region

Figure 3.5 presents the coverage percentage as a function of the last time it was achieved in the discrete case. In this configuration, the relative weight of overhead is predominant



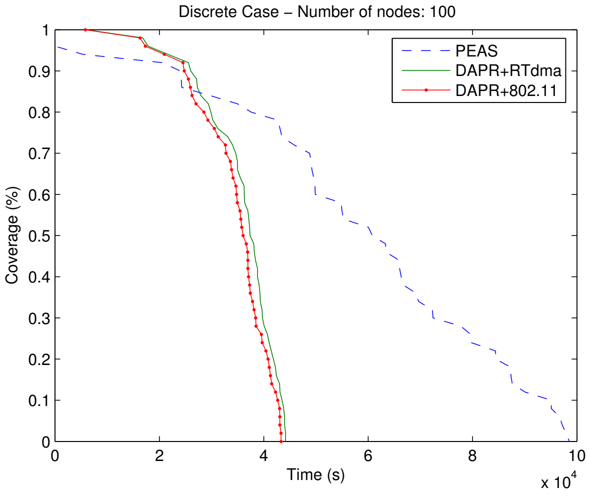
(a)

(b)

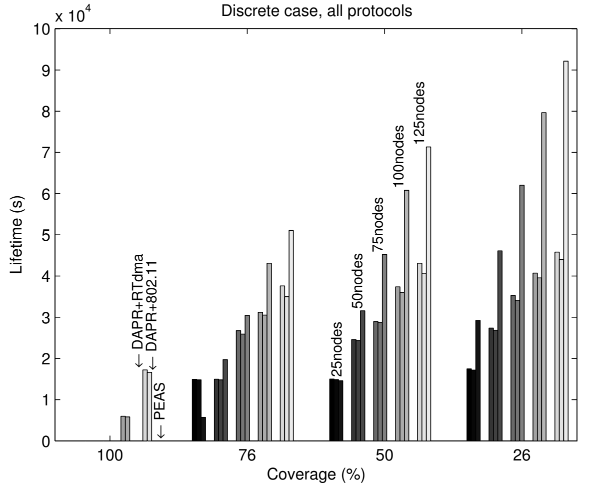
Figure 3.4: Continuous monitoring: network coverage over time for DAPR with original cost ($C_{ac}(S_i) \propto \frac{1}{\Sigma\epsilon_{rem}}$) and PEAS, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 100 nodes. For this application, PEAS is greatly outperformed, and the MAC level inefficiencies of DAPR are no longer observed.

since only one packet is sent every $10\ s$.

Figure 3.5 shows that DAPR (with both MAC protocols) provides large coverage ($> 85\%$) for longer than PEAS; however, the amount of time that DAPR can achieve



(a)



(b)

Figure 3.5: Discrete monitoring: network coverage over time for DAPR and PEAS, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 100 nodes. For this application, DAPR outperforms PEAS for coverage $> 85\%$ but the low overhead of PEAS guarantees a longer lifetime for mid-to-low range coverage.

medium coverage ($< 75\%$) is much shorter than for PEAS. For 100 sensors, 95% coverage is provided for up to $20,000\ s$ with DAPR+RTdma, $19,000\ s$ with DAPR+802.11, and $2,000\ s$ with PEAS. The RTdma MAC allows for energy savings compared to

IEEE 802.11. However, since the number of broadcast packets is much higher than that of unicast packets, the gains offered by TDMA are minimal in this case. The overhead of DAPR, although it enables a maximum coverage for a longer time, shortens the lifetime for networks that do not require large coverage.

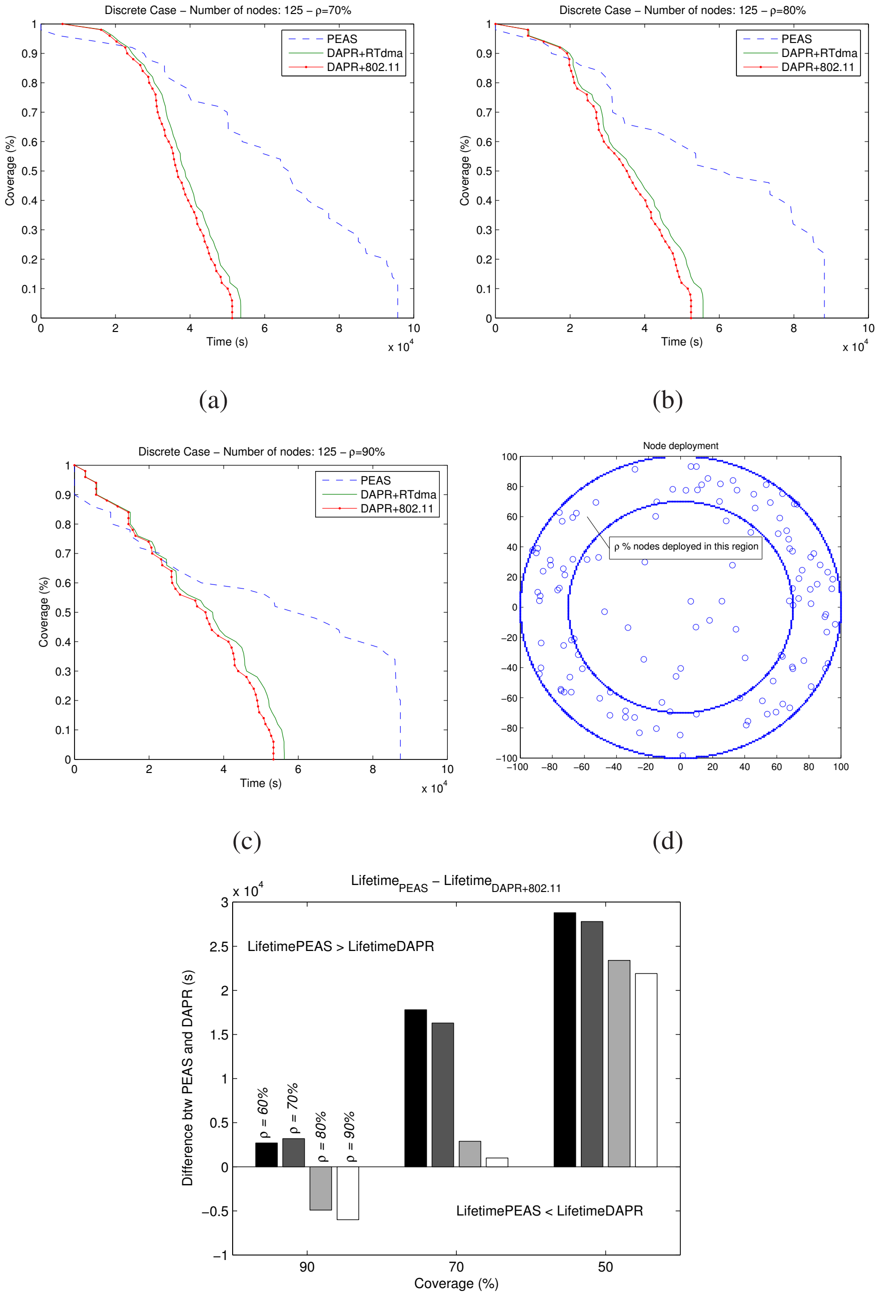
Alternatively, PEAS, capitalizing on its minimal overhead, shows a lifetime at 50% that is superior to that of DAPR. As the number of nodes increases, so does the difference observed between the lifetime of DAPR and PEAS for medium to low coverage: while the overhead remains the same for PEAS, it increases drastically for DAPR since the overhead is related to the number of neighbors.

Additionally, the small number of data packets on the network has no significant impact on the lifetime. For PEAS, the network lifetime increases almost linearly with the number of nodes, as seen in Figure 3.5(b). In effect, when only n nodes are required to provide a sufficient coverage, the remaining sensors are asleep. When the first set of nodes dies after t seconds, a new pool of sensors turns on, prolonging the network lifetime by t. In the case of DAPR, overhead penalizes the network, accounting for smaller gains in network lifetime as the number of nodes increases—thus explaining the non-linearity of coverage increase with the number of nodes observed in Figure 3.5(b).

3.3.4 Non-uniform Node Deployment

In this set of experiments, ρ percent of the nodes are placed in a region located beyond a circle of radius $70\ m$ and inside a circle of radius $100\ m$ (see Figure 3.6(d)). We ran simulations for ρ equal to 60%, 70%, 80%, and 90%. Figures 3.6(a)(b)(c) show the evolution of the network lifetime for various ρ (most nodes are located at the periphery of the network). As seen from these graphs and Figure 3.6(e), DAPR's relative performance increases as the non-uniformity of the node deployment becomes more pronounced. The DAPR application cost lets the network realize early that the sensors located in the center of the network are critical to the network coverage and thus DAPR tries to preserve them.

On the other hand, PEAS indiscriminately chooses these sensors to serve as routers (as AODV favors shorter routes). In PEAS, high coverage is rapidly lost in these non-uniform scenarios, while low-range coverage is maintained for a longer time thanks to the low overhead of PEAS. Thus, the cross-layer scheme provides a clear improvement



(a) (b)

(c) (d)

(e)

Figure 3.6: d) shows a typical node deployment for discrete monitoring (the inner and outer radii are $70\ m$ and $100\ m$). a), b) and c) are the network coverage over time for 125 nodes for 70%, 80%, and 90% of the nodes deployed in the ring. e) is the difference in lifetime ($Lifetime_{PEAS} - Lifetime_{DAPR}$ in seconds) for various ρ at 90%, 70%, and 50% coverage. As the non-uniformity of the deployment increases, so does DAPR's relative performance.

in coverage / lifetime but is countered by its own incurred overhead.

3.4 Gains of Cross-Layering Within the Same Protocol

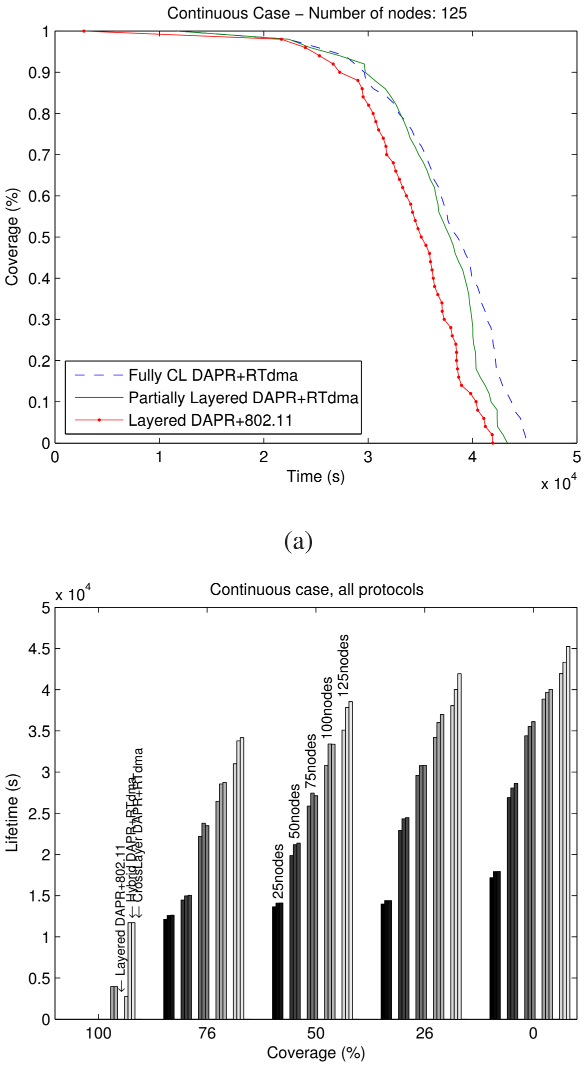
3.4.1 Advantages and Drawbacks

As the previous results have shown, there is no architecture that performs consistently better than the others for all applications and all levels of coverage. In general, cross-layer approaches are harder to create and modify: they are usually more complex, compute intensive, and require more overhead.

This being said, we can define a set of tunable parameters from cross-layer architectures that would allow for a better QoS in a real-time environment: the definition of coverage (*e.g.*, geographical coverage, covering the needs of the application such as health monitoring, signal-to-noise ratio (SNR)), the query interval (to minimize periods during which the delivery of data packets is not guaranteed or optimal such as during the route and role discovery phases or guarantee peak services), and application cost (to minimize partitioning, delays, etc.). For example, we could envision that the application cost be changed when a node outputs a large amount of data (deemed important) to avoid a hot spot problem.

The layered protocol, on the other hand, enjoys a relative simplicity, keeping the overhead to low levels. However, adaptability is not a measure of how tunable the protocol may be: PEAS provides only a small set of parameters (sleep duration, number of probe and reply retransmissions) that a user or a middleware could refine to adapt to a changing environment or application requirement. It is unclear, however, exactly what benefits cross-layering brings, or whether a smart layered design could provide similar performance.

In order to quantify the gains only ascribable to the cross-layer nature of the design, we created a layered version of DAPR. For a partially layered (or *hybrid*) DAPR, nodes elect whether or not to activate based on individual node costs, not cumulative route costs—thus the selection of active sensors and routers is done independently and with little correlation. Further layering of DAPR necessitates no longer checking usefulness of each node at the MAC layer but at the node activation layer (layered DAPR).



(a)

(b)

Figure 3.7: Continuous monitoring: network coverage over time for cross-layer, hybrid, and layered modified DAPR using the energy cost $\frac{1}{\epsilon_{rem}}$. (b) is the network lifetime for 25, 50, 75, 100, and 125 nodes. (a) is the coverage percentage as a function of time for 125 nodes. The fully cross-layer and hybrid designs perform similarly for high-range coverage, and the only significant difference happens for low-range coverage.

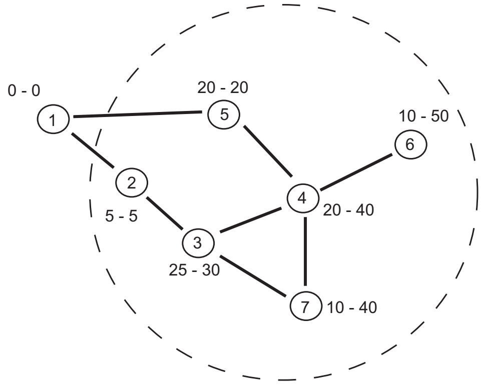


Figure 3.8: The numbers represent: *NodeId* in circles, *Individual Cost* - *Cumulative Cost*. Recall that nodes with the smallest cost forward the *query* first. If nodes 3 and 4 have overlapping sensing areas, node 4 (individual cost of 20) will activate but is part of a path with higher cost than 3 (individual cost of 25). Node 7 will thus route its packets through 4, using a less efficient route than if 3 had activated.

3.4.2 Additional Results: the Impacts of Cross-Layering

3.4.2.1 Energy Application Cost

In Figure 3.7, we first present results obtained using an individual energy cost ($\frac{1}{\epsilon_{rem}}$) instead of the original (horizontal cross-layer) DAPR application cost, as first introduced in Section 3.3. Such a cost is not a measure of how densely a subregion is covered, and consequently can be seen as an application independent feature within cross-layer, hybrid, and layered architectures.

Figure 3.7 shows that the cross-layer design performs very similarly to the hybrid one. The two designs yield equal high-range network coverage, and only for mid to low-range coverage does the fully cross-layer architecture outperform the hybrid and layered designs. The layered design (DAPR+802.11 with no MAC cross-layer optimization) does not suffer from the problem exposed in Section 3.3.2.1: the individual costs are sufficiently different that few nodes try to opt out at the same time. When the same design is used with RTdma, the simulation results are in all points similar to the hybrid case presented here, and thus we decided not to show them.

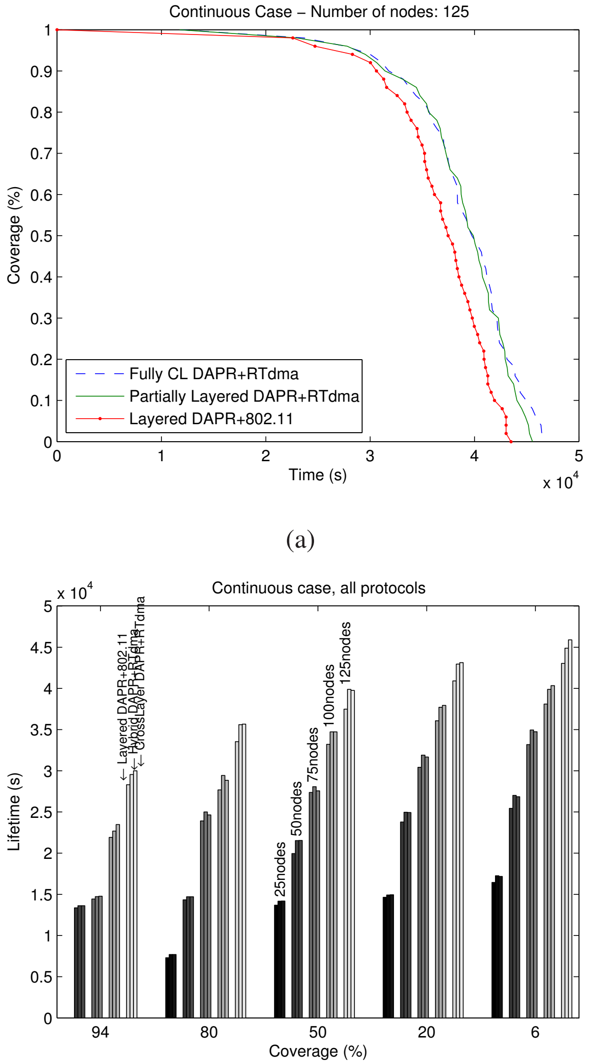
In the hybrid and layered schemes, sensors that activate tend to be the ones with the most remaining energy while their path to the sink may be comprised of intermediary nodes with very low remaining energy. Figure 3.8 illustrates this non-optimal selection mechanism. In this example, nodes use their individual (energy) cost instead of a cumulative cost. Nodes with low application cost forward *queries* earlier, thus setting paths that include sensors with low remaining energy. If sensors 3 and 4 have overlapping sensing areas, 3 will turn off, forcing node 7 to use a non-optimal path. Later in the simulation, these *bad choices* reduce the network lifetime.

At the end of the simulation, a greater disparity is observed between the cross-layer and layered schemes. The use of the individual energy cost (layered DAPR+RTdma and hybrid DAPR+RTdma) causes many nodes with very low remaining energy to consider opting out at the very end of the role discovery phase. The beacon delay is proportional to the inverse of the remaining energy and can be at most equal to the discovery phase duration. When this maximum value is reached for several neighboring nodes, the node activation process is random: their sometimes very different costs can only be represented by one value of saturation.

These results show that cross-layer gains are possible, but they remain marginal. The following subsection gives another reading of this issue.

3.4.2.2 Original DAPR Application Cost

Figure 3.9 shows the network coverage as a function of time in the continuous case with the original DAPR application cost. While modest gains from partial cross-layering exist for very-low coverage monitoring, the advantages of using a fully cross-layer design seem overall marginal. On the other hand, the fully cross-layer design retains clear advantages over the layered approach for all network coverage. However, the layered design using RTdma in place of IEEE 802.11 (not shown in Figure 3.9) performs exactly as the partially layered design since contention and queuing delays are not as critical. In this design, nodes evaluate if they have to opt out at the node activation level whereas the MAC layer has no consideration for this, but the RTdma scheme ensures that beacons are delivered with acceptable delays. The joint selection of routes and node activation offers no clear advantage over the layered approach: queries still reach distant nodes using paths of smallest cost because the delays before forwarding a query are proportional to costs. Both costs are a reflection of how densely covered a region is.



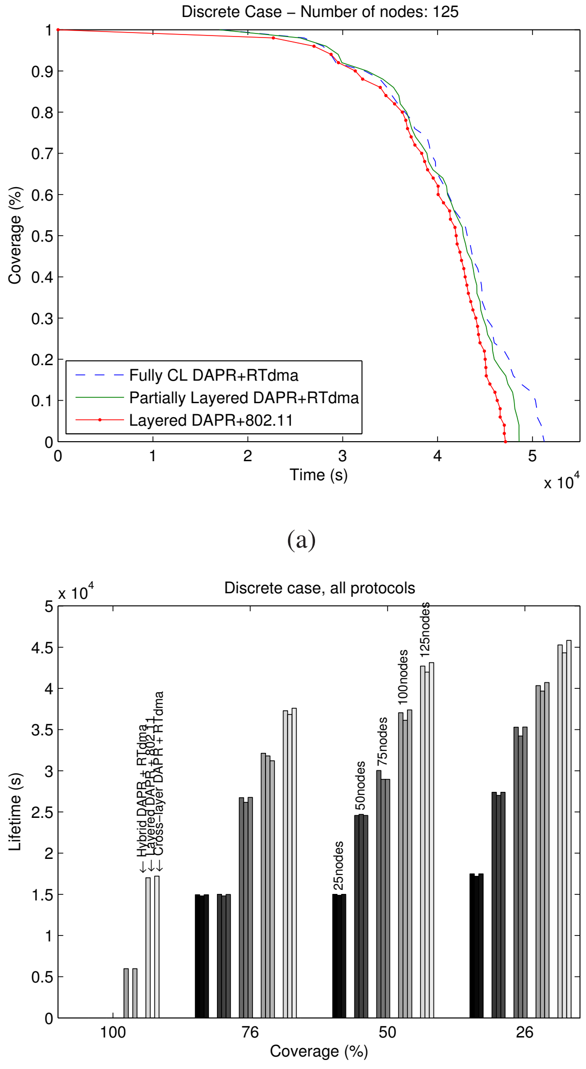
(a)

(b)

Figure 3.9: Continuous monitoring: network coverage over time for cross-layer, hybrid, and layered DAPR, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 125 nodes. The cross-layer and hybrid show no significant difference in lifetime.

As seen in Figure 3.10, the simulations in the *discrete monitoring* case provide similar results and differ only by an increase in the network lifetime (fewer packets are sent every round).

In the non-uniform deployment case first introduced in Section 3.3, the simulation



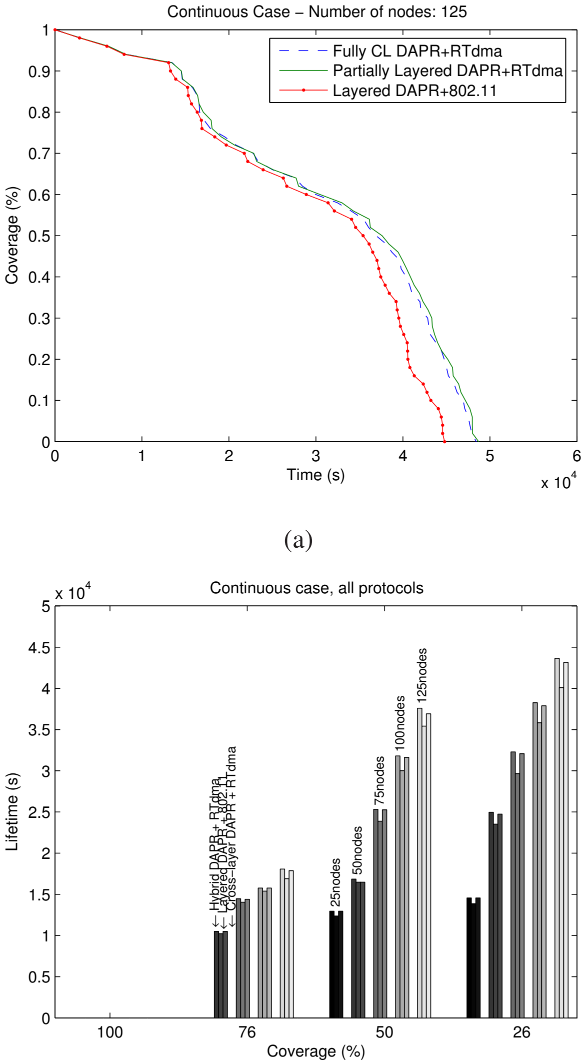
(a)

(b)

Figure 3.10: Discrete monitoring: network coverage over time for cross-layer, hybrid, and layered DAPR with original cost, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 125 nodes. No significant difference is observed between cross-layer and hybrid designs.

results presented in Figure 3.11 show that there is no significant gain obtained from cross-layering although the network topology should be more advantageous to the fully cross-layer designs.

To conclude this section, we argue that the cross-layer gains obtained when using



(a)

(b)

Figure 3.11: Continuous monitoring in a non-uniform deployment with 90% of the nodes at the periphery: network coverage over time for cross-layer, hybrid, and layered DAPR, given for 25, 50, 75, 100, and 125 nodes in (b). (a) is the coverage percentage as a function of time for 125 nodes. No significant lifetime gain is obtained for cross-layer DAPR over the hybrid scheme even though the non-uniformity should advantage the former.

the remaining energy as a cost are greatly exceeded by a more efficient cost (*i.e.*, application cost). This application cost, which brings an estimate of the density of coverage in a region, appears more significant in improving performance than cross-layer vs. layered designs.

3.5 Discussion of the Usefulness of Cross-Layering

Our work and that of others [27] have shown that cross-layering provides sometimes elusive advantages and is strongly subjected to specific application goals and protocol designs. Cross-layering can help alleviate protocol design weaknesses and limitations. For instance, PEAS fails to correctly serve the *continuous monitoring* application due to its design, which left no provision to send nodes back to sleep. One could imagine stopping packet flows around a node probing for neighbors. While this would not render the protocol immune to *probe* packet losses, the *probe* replies would be more likely to be received successfully, allowing a sensor to sleep longer. However, this is a cross-layer approach, with interaction between the sensor management protocol (PEAS) and the network and / or MAC layers, thus corroborating the argument that cross-layer protocols can, at times, better serve specific application needs.

It is also obvious that gains can be obtained when the physical layer uses location information (typically found at other layers or provided by a specific service) to control the transmission power.

Although introduced a few years ago in the research community, there are many definitions of cross-layering in sensor networks protocols. The following are two different definitions for cross-layering:

- *Information sharing*: the most commonly used, this architecture provides support for sharing specific information available at one layer to at least one other layer of the protocol stack.
- *Layer fusion*: in this architecture operations typically executed at different layers are combined in order to optimize the protocol.

Transmission power control and the MAC optimization to check whether a sensor's coverage is needed are examples of *information sharing*. Conversely, the mechanism of route determination and node activation in DAPR is an implementation of *layer fusion*. While benefits for *information sharing* are obvious and well established, *layer* *fusion* does not seem to deliver on the promised gains as specifically shown by our comparative study of the cross-layer and layered DAPR protocols. *Layer fusion* does not appear to bring significant improvements when the design of the architecture is sound.

We argue that all layers of a protocol should have access to the same valuable information through a platform reaching across the stack. The details of such a design are given in Chapter 4. It is clear however that such an architecture could aim to provide information to allow the individual protocols to optimize a set of tunable parameters. In the case of DAPR for instance, we argue that sharing information about the application requirement can greatly benefit the overall network lifetime. Also, several layers in the protocol may have an interest in knowing the application cost, the query length, the neighbors' status, etc. A simple illustration in the case of PEAS is the following: we can imagine having the node activation protocol send several probes and replies when a node detects that congestion is severe, with the exchange of information about current congestion conditions sent from the MAC layer through the proposed platform.

3.6 Summary

This chapter presented a comparison of a cross-layer architecture versus a layered approach for continuous and discrete monitoring applications in wireless sensor networks. The cross-layer architecture enables its users to benefit from full coverage for slightly longer than the layered architecture when reporting on a discrete event, but it greatly reduces the lifetime for smaller coverage. The overhead of the cross-layer architecture becomes limiting in the face of so little traffic on the network, but this overhead is necessary to obtain high coverage in an evolving network.

For continuous monitoring by all active nodes, the resilience of the cross-layer design preserves the full coverage by shifting the weight of relaying packets to sensors placed in dense areas. In this case, the cross-layer architectures greatly outperform the layered design. While hard to quantify, each level of integration in the cross-layer protocol adds to the lifetime of the network as a whole. Our work extensively shows that while cross-layer designs are not the panacea to all applications, they are able to better serve applications with specific requirements (for instance, a requirement of 95%

coverage for as long as possible).

When comparing two configurations of the same protocol (layered and cross-layered DAPR), we observed that the architecture benefits from sharing information, but we were unable to link a high degree of integration between two layers to any significant improvement in the network lifetime. Thus, we propose creating an architecture that maintains a layered design, but where all the protocols in the stack would have access to important information regarding the state of the sensor, network, application, etc.

While obviously not exhaustive, our simulations provided a data point in the otherwise large and uncharted domain of cross-layer improvements and trade-offs. It remains clear that light-weight protocols such as PEAS can outperform more involved cross-layer designs like DAPR. However, the current trend of sensor network implementations is cautiously shifting toward serving more complex applications, for which the former may be too simplistic. Efficiently sharing information in cross-layer designs will greatly counter-weight the inevitable increase in overhead.

This work shows that DAPR may adapt to the (possibly changing) requirements of an application through a smart middleware. Information from the application may be used to tune internal parameters to save precious nodes for critical moments of the network deployment. A set of tunable parameters of the cross-layer design (such as round length and application cost) can be defined. Another teaching of this study reveals that an efficient new sensor network architecture would be one that shares relevant information with all the layers of the protocol stacks.

Chapter 4

A New Cross-Layer Information-Sharing Sensor Network Protocol Architecture

4.1 Introduction

While protocols and algorithms for wireless sensor networks (WSN) have been the subject of much research, little attention has been paid to sustainable architectures for these networks. The use of sensor networks in every day life has been slow in coming, as sensor networks are still subject to enduring challenges in energy conservation and bandwidth use, as well as the lack of a durable and flexible yet supportive architecture. Historically, little thought has been given to specific architectures for wireless sensor networks, with the most widely used architecture inherited from wired computer networks in the form of the OSI model.

To date, there have been numerous proposals of cross-layer protocols, which use the specificities of sensor networks to improve the network lifetime and the response to the end application. The word “cross-layer” may refer to various designs, many of which are identified by Srivastava et al. in [26]. One type of cross-layering fuses two or more layers into a single, integrated layer (Figure 2.1(b)). Another type of cross-layering allows information to be shared among several (non-adjacent) layers (Figure 2.1(c)).

Clear advantages of cross-layer designs include the following:

- They provide a network and application specific response to the user's needs by closely adapting the protocol stack to the requirements and constraints of the deployment.

- They can greatly improve the network's performance and lifetime.

- They help rid the stack of unnecessary layers of the OSI model in some deployments.

These advantages come at the cost of the following drawbacks:

- Cross-layer designs are hard to maintain and update, and they make it difficult to replace parts of the protocols.

- They require a coherent design for real gains and to avoid unforeseen and undesirable interactions [62] [63] [27].

- They cannot be ported to other applications since they lack generality.

To have a chance at sustaining the development of wireless sensor networks, a protocol architecture should exhibit the following desirable characteristics, among others:

- Flexibility: new protocols or improved versions of existing ones should find their way into current WSN deployments or developments in order to gain from the latest technology. However, this benefit may be outweighed by the amount of work necessary to include them into an existing framework. A strong WSN architecture should thus ease protocol swapping.

- Information freshness: protocols often rely on a set of parameters that define a local sub-network at a certain time. It is desirable that an architecture provides an up-to-date vision of a node's neighborhood.

- Simplicity: In order to guarantee the quick adoption of WSN as a preferred solution for industry, an architecture should be simple to use. We believe a simple architecture is one that does not carry hidden operations and that provides a unified access to its data structures.

At the same time, the architecture should enable the protocols to achieve long network lifetimes for various applications. These are, in general, conflicting goals, with the former set of goals achieved using layered architectures and the latter goal achieved using cross-layer architectures.

Interactions between layers, as enabled by cross-layer designs, make protocol maintenance and replacement a complex and time consuming task. In the research field of wireless sensor networks, protocols are introduced and improved regularly. Because wireless sensor networks are as diverse as the applications they serve, no one protocol suite has emerged (nor likely will in the foreseeable future) as a universal solution. Consequently, new WSN architectures are needed that ease protocol substitution and maintenance, and this is one of the goals of our work.

Following the seminal work of Kawadia et al. [27] urging researchers to adopt a cautionary stance on cross-layer protocols, many use great care—if not a dose of skepticism—in designing cross-layer schemes. One solution to prevent the *spaghetti design* mentioned in [27] is to retain a traditional layered structure in the protocol stack but share information among the layers, as promoted in [64]. While such architectures do not eliminate the need for careful protocol design, they can guarantee the availability and correctness of information shared among many levels of the stack.

The first steps taken towards such an information-sharing architecture occurred in the field of mobile ad-hoc networks (MANET) with MobileMAN [37] and CrossTalk [39]. Later, SNA [40] was introduced specifically for wireless sensor networks. Chameleon [42] shares many of the goals of SNA but provides a different solution. XLM [43], proposed by Akyildiz et al., takes the interesting opposite solution and fuses all communication layers to best support the sensor network application. These architectures present a variety of approaches to meeting the goals of flexibility, universality, simplicity and support for application goals.

In the previous chapter, we showed that while *information sharing* can be beneficial, *layer fusion* shows surprisingly little improvement in the face of other design optimizations. We therefore propose in this work an architecture that provides support for the former, while still leaving open the possibility for the latter.

The most common existing protocol architectures are the OSI protocol stack and application-specific cross-layer architectures. The layered stack does not provide a common framework for information sharing and thus cannot provide maximum lifetime for sensor network applications. Application-specific cross-layer architectures, on the other hand, while greatly improving network lifetime, lack the generality that would help popularize wireless sensor networks and ease their development for different applications. Furthermore, no existing protocol architecture supports protocols that require information not typically ascribed to that layer's primary function. For example, oftentimes node location is required at the network layer, which should be solely responsible for routing data throughout the network and not finding node locations. Thus, outside *services* should provide this extra information that may be needed by one or more protocols.

The contributions of this chapter are twofold: we survey the above-mentioned architectures, exposing their relative strengths and weaknesses against a set of desirable (and at times, contradictory) goals, and we propose a new architecture called X-Lisa (Cross-Layer Information Sharing Architecture) that maintains a layered structure for increased generality, while supporting cross-layer information-sharing. X-Lisa provides information repositories for storing the sensor node's current state and local view of the network. Access to, and freshness of, these tables is guaranteed to all layers through a fixed interface. Additional support for the protocols comes from a library of services available to the programmer. X-Lisa provides support for the discovery, maintenance and sharing of critical information, thus allowing protocols to re-focus solely on their primary tasks. This eases the burden of designing new protocols by allowing the protocol designer to pre-suppose the existence of important information without requiring the protocol to directly find this information. Our implementation of X-Lisa in TinyOS includes elegant solutions to guarantee flexibility even in this simple and stripped down operating system. At the same time, this common framework for information-sharing comes at a cost of an increase in storage overhead and packet sizes, and thus it is not suitable for extremely resource-constrained sensor networks.

4.2 Architectural Approaches

The widespread success of the OSI model [65] in wired networks provided a starting point in architectural design for protocol stacks in the new fields of MANET and WSN, with the OSI model constituting the legacy architecture for these new, more complex networks. Although the OSI model had not been intended for the specificities of WSN and MANET, this architecture proved flexible enough (with the help of some ad-hoc violations to accommodate cross-layering designs) to stimulate the growth of these two fields. In this traditional architecture, direct communication is only permitted between adjacent layers. Protocols may obtain information from the packet headers and data units, and from the same layers in distant or direct neighbors. This incurred more horizontal communication (between neighbors), which the bandwidth of wired networks could easily support. However, cross-layer protocols have introduced architectural violations that allow a layer to communicate in some form or another with non-adjacent layers. Such designs aim to benefit from the specificities of WSN to increase network lifetime and application quality of service (QoS) support, and to avoid wasting bandwidth, a scarce resource in WSN.

Existing architectures MobileMan, CrossTalk and SNA and our proposed architecture X-Lisa all fall into the shared databases described by Srivastava [26] by defining a set of commonly available data structures. XLM represents an example of merging of adjacent layers.

Figure 4.1 presents a taxonomy of the architectures surveyed in this work. We focus on the architectures that allow information-sharing through a shared database. We also include XLM in our survey because it supports cross-layering while taking the counterpart of information-sharing.

4.2.1 MobileMan: Subscription to an Abstracted Database

MobileMan [37] provides an abstracted database, called the network status (NeSt), which is made available to all layers of the stack through a publish / subscribe API. NeSt organizes the exchange of information in the stack: a protocol that needs information from other layers has to register with NeSt and subscribe to event notifications regarding this information. Protocols whose information is of interest to others must notify NeSt of the occurrence of an event. NeSt then delivers the incidence match to the various protocol subscribers.

To the best of our knowledge, NeSt does not organize information exchange about a node's neighbors (horizontal information-sharing), and it leaves the burden of event

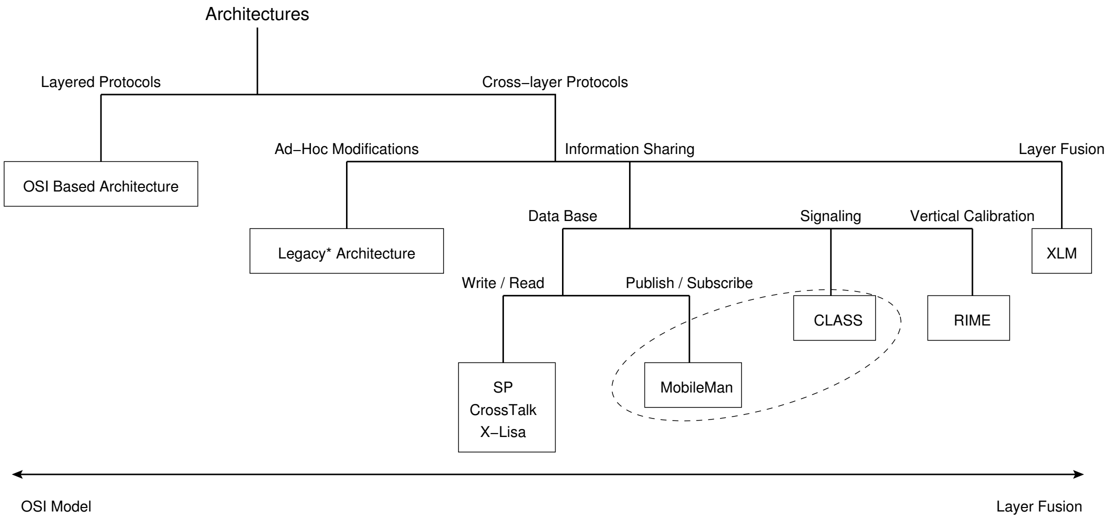


Figure 4.1: Classification of some architectures for MANET and WSN. MobileMan abstracts a data base through signaling, as highlighted by the dotted circle.

notification to data-supplying protocols. Adding such functionalities to all the protocols in the stack may involve levels of refinement and complication not suited for MANET, and *a fortiori* WSN. This is because the goal of MobileMan is to adapt Internet legacy protocols and applications to MANET, where the benefits of cross-layering are greater. While MobileMan facilitates protocol swapping, the end goal is to support Internet technology over wireless ad-hoc technologies.

The solutions retained by the MobileMan workgroup are elegant and ingenious: NeSt does not store data but merely provides an abstraction for information exchange. It also allows complex events to be disambiguated and reported to the protocols that have an interest in them. However, NeSt asks much of the protocols in the stack, possibly causing information exchange to be hindered by very procedural access.

4.2.2 CrossTalk: A Common Database for MANET

Because publish / subscribe mechanisms may be complex and lack generality, a non-abstracted database made available to all layers in the stack may be more suited to the needs of networks with limited resources. CrossTalk [39] exemplifies how such a database may be used and populated.

The goals of CrossTalk are quite similar to those of MobileMan: adaptability and flexibility are desirable to allow ease of protocol maintenance and replacement. CrossTalk provides local information to all protocols in the stack, as well as a *global* view of the network: a set of parameters deemed of interest to the protocols is gathered about distant nodes. A global view of the network is thus propagated, forming a context for each node. With this information, individual nodes can evaluate their relative situation and modify their behavior accordingly. For instance, a back-off value of two seconds at the MAC protocol of a node has little meaning by itself, whereas knowing that the average back-off in the visible network is $1\ s$ indicates over-utilization.

Because it may represent considerable amounts of raw data, the information received by every node is compounded, possibly using a weighted average. Generally speaking, information from very far nodes receive small weights, as well as that of very close nodes (because of inherent local correlation), giving the averaging weights a triangular shape. A similar weighting technique can be applied in time, because very old information bears little importance.

The global view of the network is formed by propagating information over several hops: a source node appends the set of parameters that are then read by every relay node on the path. Winter et al. argue that a global view of the network at the individual nodes can greatly improve the performance of the network. However, the implementation of CrossTalk was carried out in NS-2, a rich programming language. Under TinyOS [2], the incurred per-packet overhead can be (conservatively) broken down into the following: $(2+2)B$ for X and Y location, $8B$ for time stamping, and nB for the value of the exchanged information.

CrossTalk seems particularly well adapted to MANET: the information propagation model takes advantage of multiple $\{$source; destination$\}$ tuples. During their evaluation, Winter et al. allowed all 200 nodes in the network to transmit data to random destinations chosen every $10\ s$. For WSN however, packets tend to travel along routes that converge towards the same data sink, and in general the same (centripetal) direction. WSN have a propensity to display characteristic bursts of packets (when an intruder is detected or during high stress states of a monitored entity), and few applications require a constant flow of packets from large subsets of the network. Thus, the amount of disseminated information can be expected to be small.

4.2.3 SNA: Abstraction of Lower Layers and Basic Functions

Culler et al. proposed the Sensor Network Architecture (SNA) in [41] then [40] with the main objective to provide greater modularity to sensor hardware designs and communication protocols. The suggestion made in [41] was to decouple aspects of the software from the underlying hardware in order to abstract the platform on which the network stack is set. To do so, the Sensornet Protocol (SP) bridges the link and network layers by abstracting key parameters of the lower layers such as link quality and scheduling information. In the next step [40], Culler et al. identify common functionalities to encourage code-reuse and runtime function sharing. SNA breaks the network layer down into reusable communication modules.

SNA retains a layered structure, providing detailed information to the various network protocols present on a node. Two predefined structures within SP serve as information repositories: the "Neighbor Table," which maintains information about direct and relevant neighbors, and the "Message Pool," which allows protocols to request message transmissions.

The neighbor table allows several protocols to share information otherwise maintained in redundant structures. A neighbor table entry usually consists of the neighbor address or *ID*, link quality, and scheduling information. Because the number of neighbors may be greater than the number of existing spots in the table, SP implements an advanced table maintenance scheme: before adding a new neighbor to the list, SP polls every protocol. If at least one protocol agrees, the node is inserted. Protocols are also notified when a neighbor is evicted from the table. Since SP is a protocol, its actions go beyond that of a simple architecture: if the power management schedule of a listed neighbor is expiring, SP asks the network and link layers to provide a new schedule. It is not clear whether the neighbor table maintains information about the node itself.

The message pool contains pointers to messages, the number of packets in a burst (for instance, in a video streaming application), notification when the next packet should be instantiated, and *urgent* and *reliability* bits. Packets are thus stored either in the data link or in the network protocol layer. SP also makes decisions about scheduling transmissions: it determines when it is most appropriate to send packets. After a transmission is complete, SP requests the next packet in the burst. Other decisions made by SP include overriding existing power management decisions to send urgent packets and batching packets in some cases.

Arguably, SNA is a considerable leap forward in WSN architectures, as it provides an unsurpassed level of flexibility and universality under TinyOS. The interface between the various network protocols and data links is rich and advanced, but also quite complex. This may hinder fast deployment of WSN. Additionally, SP does not propagate neighbor information automatically and solely relies on protocols to populate the neighbor table (although SP may post requests regarding table information freshness). As a whole, SNA tries to achieve a different goal from ours since we are more concerned about structure reuse rather than code reuse.

4.2.4 Chameleon: Abstraction of Communication Protocols

The main goal of the Chameleon architecture [42] is to abstract communication layers so that WSN protocols may run over any network, from 802.15.4 to IP. Abstraction is achieved thanks to packet *attributes*, an abstract representation of information contained in packets. Rime, a layer contained within Chameleon, takes care of mapping attributes to any standard header.

Additionally, cross-layer interactions are supported through “vertical calibration”: information is contained in the *attributes* (header) of packets that are passed between layers. However, in order to propagate information to all layers, the packet headers are not removed after being processed.

Like SNA, the ultimate goal of Chameleon is to provide abstraction to lower layers by identifying basic protocol primitives, although Dunkels et al. selected different ones from SNA. It differs from our intention to allow cross-layer interactions between protocols, since information is not shared among all layers in the stack.

4.2.5 XLM: The Counterpart—Fused Layers

Akyildiz et al. proposed XLM [43] as a fused-layer module, regrouping all protocols from the data link to the node activation layers. Because this organization of the protocols has a redefining impact on the larger architecture, we consider XLM as an interesting counterpart to simple information-sharing.

The principle of node communication in XLM is *initiative determination*: after a node indicates it has a packet to send with an RTS, each neighbor decides whether to participate in the communication based on a set of rules whose metrics and parameters are issued and maintained by various functions in the module. Let $\mathcal{I}$ be the initiative:

$$\mathcal{I} = \begin{cases} 1 \quad \text{if} \begin{cases} \xi_{RTS} \geq \xi_{Th} \\ \lambda_{relay} \leq \lambda_{relay}^{Th} \\ \beta \leq \beta^{max} \\ E_{rem} \geq E_{rem}^{min} \end{cases} \\ 0 \quad \text{otherwise} \end{cases} \tag{4.1}$$

These parameters set conditions on link state, flows, buffer fullness, and energy. If all conditions in (1) are satisfied, the node participates in the communication; otherwise it goes to sleep until the next determination time. According to the authors, this set of conditions ensures reliability of the link, manages flows and buffer levels, and guarantees uniform energy consumption.

Akyildiz et al. argue that the tight meshing of all functionalities guarantees optimality of the network response—or at the very least the coordination of their actions. All protocols fused in XLM benefit from the same up-to-date information, much like the previous approaches. However, XLM supposes that CSMA/CA is the “best” MAC protocol and builds off of this assumption. Whether this supposition is correct or not is beyond the scope of this survey, but the fused nature of the protocol layers renders protocol swapping a complicated process. Unlike MobileMan, CrossTalk, and SNA, there is no standardized interface between the various protocols of the module. If a low-power-listening MAC protocol such as X-MAC [21] were to replace CSMA/CA, a programmer would have to learn about the entire module in order to replicate the socket with other functions.

4.3 Architecture Comparisons

In this section, we present a qualitative comparison of the architectures described in the previous section. It is expected that the architectures will perform better in some regards, and more poorly in others, in accordance with their different design goals. The aim in this section is to provide a side-by-side comparison of each architecture’s relative advantages and drawbacks.

Table 4.1: Qualitative comparison of existing architectures (given without order of importance).

	Legacy *	SNA	Cham.	X-Talk	MobMan	XLM
Modularity	—	+	+	+ / ?	+	—
Universality	—	+	+	+	=	—
Event Notification	—	—	—	—	+	—
Service Support	—	=	=	—	—	—
Table Maintenance	—	+	N/A	?	N/A	N/A
Information Prop.	—	—	N/A	—	—	N/A
Overhead	+ (small)	+	+	—	—	+
Simplicity	+	—	+	+	—	+

Table 4.1 presents the strengths and weaknesses of each architecture for a specific set of goals. The term *Legacy** refers to the layered structure inherited from the OSI model, with potential ad-hoc modifications (“violations”) to support cross-layering.

4.3.1 Flexibility

This section evaluates the *flexibility* of an architecture, and thus its chance of becoming popular. In WSN, the term flexibility indicates both modularity and universality, which are often contradictory in nature, as modular designs have, so far, denied support for cross-layer protocols.

4.3.1.1 Modularity

Migration from strictly layered to cross-layer designs has incurred architecture violations that make swapping protocols an intricate task. Modular designs guarantee that replacing a protocol requires little more than inserting the new protocol and using the appropriate interface to connect the layers. This is accomplished in TinyOS via a simple rewiring.

It is agreed that the OSI model adapted to WSN (the *Legacy** architecture) is the most modular architecture. However, this architecture is unfit for cross-layer protocols, which introduce protocol-dependant violations. The legacy architecture rates poorly in modularity because of the unforeseeable nature of these violations.

SNA, Chameleon, CrossTalk and MobileMan are all modular, although to varying degrees. Because SNA's and Rime's goals are to abstract lower layers, modularity may not apply to higher levels in the stack. We believe, however, that SNA is the most modular architecture to date proposed for WSN. The implementation details of CrossTalk are not provided in [39], and thus, we can only base this evaluation on what CrossTalk sets out to do. CrossTalk does not appear to keep packet information in a message table, which may hinder protocol replacement. Because sharing information between protocols requires writing additional access functions, MobileMan is not as modular as other designs, and to the best of our knowledge, does not support the sharing of neighbor information. Lastly, since XLM focuses only on network performance, replacing a function in XLM requires knowledge of the full module, making it low on the scale of modular designs.

4.3.1.2 Universality

The term *universality* refers to the ability of the architecture to accommodate all platforms, protocols, and end applications. SNA and Chameleon propose an elegant solution to abstract the underlying hardware from the protocols. CrossTalk offers a good solution for MANET, however the information propagation model is not suited for WSN. Depending on the application, only a small subset of the nodes in a WSN send packets to a very limited number of data sinks. Thus, the information contained in packets sent over these routes cannot supply a global view of the network, except possibly close to the data sinks. Additionally, as for MobileMan, there are no provisions to abstract the data link layer, which heavily depends on the platform. Finally, changing a function within XLM requires the knowledge of the full module and its subsequent modification.

4.3.1.3 Event Notification

One of the advantages of layer fusion is the ability to coordinate various functions easily: protocols now gathered under the same layer can share information and invoke the same set of functions. A layered design, however, is required to create more complex interfaces between different layers, which contradicts the goal of modularity. For instance, imagine a MAC protocol able to quickly detect broken links and a routing protocol whose route repair mechanisms are slow. A programmer may wish for the MAC protocol to notify the routing protocol of a failed link as early as possible. Two solutions are possible: the layers may be fused, or the interface between the two layers may include a command to start repairing the route immediately. This makes replacing either protocol a harder task as a programmer would have to learn about the inner workings of both layers. We argue that an architecture should allow event notification through its shared database. Of the surveyed architectures, only MobileMan supports event notification.

4.3.1.4 Service Support

Rarely mentioned, services provide key support to protocols in the stack. For example, services may gather information about the remaining energy of a node, the location of a node, etc. Protocols should only focus on their main role, such as routing, without concerning themselves with gathering peripheral information. To the best of our knowledge, no existing architecture proposes organizing services in their stack.

On top of managing services, the architecture should allow individual protocols to selectively load them before deployment, and to turn them off during runtime.

4.3.2 Information Freshness

In this section, we examine how the network view is kept up-to-date, a key feature for many protocols. This feature is critical to having an architecture that is modular and easy to maintain and use. Maintaining up-to-date information requires intelligent table maintenance and information propagation.

Most architectures propose to store information made available to part or all of the layers in the stack. The architecture should remove stale entries from its data repositories, while making sure that these repositories do not exceed full capacity. However, we believe the architecture, *not* the individual protocols, should manage the neighbor table. Conversely, in the case of the message pool, the **Send** interface in TinyOS provides a `sendDone` function whose implementation can easily manage packets in the message pool.

SNA (SP) features advanced table maintenance, although more decisions are left for the individual protocols to make. Because we lack implementation details for CrossTalk, we cannot evaluate it in this regard. Chameleon does not provide a common information repository, and thus does not provide a local view of the network. Because MobileMan abstracts information structures, there is no actual common neighbor table, and information freshness and management is handled by the registered protocols. Finally, XLM does not need to make any information structure available to other layers in the stack because it already shares common data within all functions of the module.

In order to maintain relevant information for all the protocols, the (global) view of the network should be propagated regularly. Only CrossTalk proposes disseminating information. However, because the main focus for CrossTalk is MANET, this propagation model is not fit for WSN, which they tend to have smaller traffic rates and converging routes.

4.3.3 Overhead

Low additional overhead is a critical notion that may qualify an architecture as the most energy efficient, and thus the most appealing. The overhead incurred by an architecture usually depends on the sophistication of the protocols in the stack. However, since MobileMan abstracts a database, redundant structures may still exist at different layers. It follows that MobileMan, like Chameleon, may require more RAM than necessary. The propagation model of CrossTalk can be viewed as expensive: at least $12B$ must be appended to each packet for X and Y locations and the time stamp. SNA and XLM provide RAM savings by removing redundant information: SNA uses a common database approach, while XLM lets all functions in the module view the same data.

4.3.4 Simplicity

A simple architecture has a greater chance at allowing faster release and deployment of WSN. A good measure of the simplicity of an architecture is its modularity. However, beyond this consideration, architectures that do not carry protocol operations seem to be the simplest to maintain and understand because they do not have unpredicted behaviors. In TinyOS, simplicity also means fewer wirings and hidden capabilities.

4.4 X-Lisa, a New Architecture for Cross-Layer Information Sharing

Armed with a new sense of the strengths and weaknesses of existing architectures for WSN, we argue that architectures relying on a non-abstracted shared database are both simple and flexible. Among them, SNA seems the most appropriate to WSN, but it does not actively fill the neighbor table, and more generally, its goal is to abstract the hardware platform from the protocols. Instead, what is needed is an architecture that provides support for cross-layer protocols using a modular structure.

Thus, we propose X-Lisa, a new Cross-Layer Information-Sharing Architecture that combines simplicity with support for cross-layer interactions, services, information propagation and event notification. In this section, we fully describe our information-sharing architecture, while in the following section we detail the improvements brought by X-Lisa.

4.4.1 The Need for a New Architecture

Our goal in creating X-Lisa is to support the exchange of fundamental information that is beneficial to all protocols within the stack, and at the same time to create an architecture that is compatible with existing and future protocols, both layered and cross-layer. To this end, we have attempted to identify a basic list of important parameters necessary for improving the performance of many protocols. These include but are not limited to:

- Node location.
- Node remaining energy (ϵ_{rem}).
- Compute resources such as CPU load, RAM use, and remaining storage capacity.
- Sensing abilities such as the set of variables the sensor can monitor.
- Contention around the node.
- SNR or probability of error (P_e), reflecting the quality of the link to any neighbor.

These node and network features may be required by one or more protocols, and oftentimes they are not straightforward to obtain. For example, while determining a node's compute resources may be simple, determining current link SNR, contention around a node or node location require specialized *services* to find this important and possibly time-varying information. In current architectures, the individual protocols are responsible for obtaining such information, and oftentimes these services are replicated in different layers, or a cross-layer fusion design is used so that the information is readily available to multiple protocols in the stack. We believe a better architecture is one that provides a common framework for obtaining such information and enabling all protocols to have access to it.

Because some protocols require only part of this information or because they use other parameters, the neighbor table should be made more flexible.

4.4.2 A New Unifying Architecture

We propose a new protocol architecture called X-Lisa, shown in Figure 4.2. X-Lisa retains a layered structure such that each layer is matched to a communication function in order to maintain a practical and simple design. While fused layer design is still possible with X-Lisa, it is not favored as it hinders modularity.

The layers are: Physical, Medium Access Control (MAC) / Data Link, *CLOI* or *Cross-Layer Optimization Interface*, Routing, Transport, Node Activation, and Application. Other functions useful to the global communication scheme can be linked to services with a specific position in the protocol stack, as will be discussed in detail in Section 4.4.7.

The Cross-Layer Optimization Interface (*CLOI*) provided by X-Lisa offers indirect access to a repository of information that may be needed by one or more protocols. *CLOI* maintains this information through three structures, a neighbor table, a sink table and a message pool, described in detail below, and it supports *services* that will fill these data structures either once or continuously, depending on the information. X-Lisa also supports event notifications to ease coordination between various layers in the stack.

While all layers and services in the stack have access to the interface, a *CLOI* layer was placed between the routing and MAC layers for two reasons. First, its location allows the interface to retrieve much of the information sent from the node onto the network as well as many incoming packets. This is useful as it allows *CLOI* to directly obtain information needed to fill the neighbor and sink tables without going through the

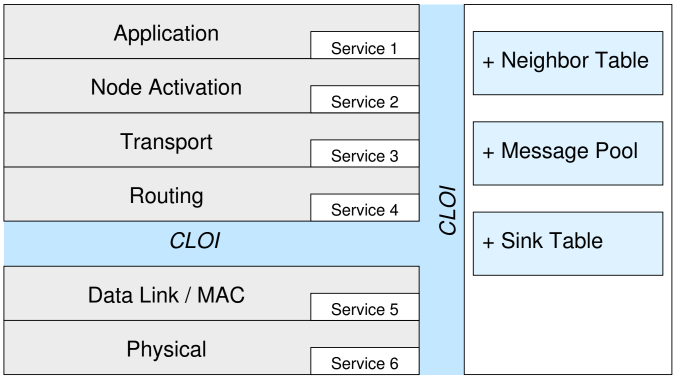


Figure 4.2: X-Lisa: An information-sharing sensor network architecture for cross-layer optimizations. This architecture retains a layered design while providing flexible information repositories as well as services to support the protocols.

protocols. The MAC and physical layers do not have a global vision of the network and cannot provide enough information about its state for automated use with *CLOI*. The second reason is that it offers the potential for abstraction of the link layer as suggested in [40].

Finally, *CLOI* has no authority to make any routing, node activation or medium access decisions, or packet reordering. *CLOI* simply acts as an interface to the protocols in the stack, allowing them to access common yet important information about the node and its neighbors that can be used to optimize each protocol's performance.

4.4.3 Information Sharing Structures

In order to support this information-sharing architecture, we need to determine the best data structures for storing the required information, and the *services* that will populate them. We propose to use three different data structures to capture all the relevant information: a neighbor table, which stores information about the node and its neighbors, a sink table, which keeps track of the various data sinks in the network, and a message pool, which stores information about current packets waiting to be transmitted by the node. Today's platforms often run TinyOS, a simple operating system for embedded targets, and that limits the scope of implementation solutions.

Table 4.2: A neighbor table is kept at every node i with non-predetermined fields. It keeps information about the node itself (for vertical cross-layering) and each of its neighbors j (for horizontal cross-layering).

ID 2B	Time Stamp 8B	n byte Array nB
Id_i	t_i	$B_{i,0}B_{i,1}...B_{i,n-1}$
1	*0x7522*	*0xAA 42 00*

4.4.3.1 Neighbor Table

Since some protocols require knowledge of differing parameters, the neighbor table is implemented as a flexible information repository. Before runtime, each programmer may elect which parameters populate the table according to the needs of the protocols in the stack. Accordingly, the neighbor table is in fact defined as three fields, illustrated by Table 4.3: node *ID*, a time stamp for data freshness, and an array of integers. The last structure may be filled with a customizable set of parameters that may or may not have the same type or size. Some of the previous solutions proposed a neighbor table with fixed fields: since NesC does not support dynamic structures, extending the neighbor table included rewriting some of the definitions and functions, and replacing fields one by one.

In X-Lisa, the programmer needs only to declare an enumeration of the fields of interest and their size in bytes. This Key-Length-Value solution allows rapid modification of the neighbor table, without resorting to changing the fields “by hand”. Although similar in spirit to Hood, it differs in its ability to update itself (with permission from- but no direct supervision by- protocols) and in its implementation. Where Hood creates several vectors of a fixed type (`int` for light, `float` for another sensor reading) of size the maximum number of neighbors, *CLOI* keeps a unified neighbor table. It is as if one was the transposed matrix of the other. This allows *CLOI* to retrieve information from a neighbor with only one pass, whereas Hood can return neighbor IDs when searching for information about a variable (which was its main goal).

By default, the neighbor table is comprised of the elements presented in table 4.3:

Table 4.3: The default fields of the neighbor table.

ID	Time Stamp	Loc.	ϵ_{rem}	Abili-ties	Entity	LQ	Status
2B	8B	4B	1B	1B	1B	1B	1B
Id$_i$	t_i	x_i, y_i	ϵ_i	$V_{i,m}$	E_i	0	S_i
Id$_j$	t_j	x_j, y_j	ϵ_j	$V_{j,n}$	E_j	$LQ_{i,j}$	S_j
1	*0x7522 (29.9s)*	*20, 50*	*0.5*	0x2 (*Light*)	0x37 (*Door*)	0.2	0x1

node *ID*, time stamp, 2D location, remaining energy, sensing abilities, monitored entity, link quality, and On/Off status. However, dynamic memory allocation is a resource consuming operation in TinyOS due to the limited available memory on the platform. Without dynamic structures, buffer space must be allocated even when the neighbor table is almost empty. Generally, a neighbor table can have up to 30 entries. When the table is almost full, retrieving information takes longer.

Because they are not predetermined, the fields of the neighbor table must be opaque and accessed through a fixed syntax, which provides read / write commands with a value and a field identifier. This process is illustrated by the following equivalent examples:

```
cost = *(float*) call
            Cloi.extractValue(entry, COST);
cost = *(float*)VALUE_ENTRY(entry, COST);
```

where `entry` is a neighbor table entry, and `COST` designates a user-defined protocol metric.

Some of this information must be updated continuously whereas other data only require infrequent updates. For instance, a node's remaining energy is a critical piece of information that changes every instant of the runtime, whereas the node's location remains fixed in static sensor networks. Trade-offs must be made in determining how often to update the fields, as updating the information too frequently incurs a large overhead penalty but updating the information too infrequently may reduce the usefulness of the data to the protocols.

Table 4.4: A sink table is kept at every node with information about each sink j in the network.

ID	Locat.	Num. Hops	Interests	
			Entity	Variables
2B	6B	1B	1B	1B
Id$_j$	x_j, y_j, z_j	N_j	E_j	$V_{j,m}$
1	*20, 50, 4*	2	0x37 (*Door*)	0x1 (*Temp*)

4.4.3.2 Sink Table

Many protocols require critical information about the various sinks in the network to determine equivalencies between them or what data to send to a particular sink. The sink table stores limited information about the various known data sinks in the network, such as the following:

- Sink ID: this is a number or description that *uniquely* identifies a sink in the network.
- Distance: this is the estimated geographical distance from the node to the sink.
- Number of hops: this specifies the minimum hop count from the node to the sink.
- Data interest: this describes the interests expressed by each sink in terms of entity and sensing variables.

Table 4.4 shows an example of this structure. Many protocols require this critical information to determine equivalencies between sinks or what data to send to a particular sink.

Because we expect information about the sinks to change slowly, the sink table is not automatically updated. Maintenance must be done by the protocols (generally a middleware and the routing protocol). It is also the only structure carrying information more than one-hop away.

Table 4.5: A message pool is kept at every node.

PacketID 2B	Description 1B	Priority and burst 1b + 7b	Status 1B
PID_1	XML tag or integer	P_1 - N_1	S_1
102	*Route Repair*	*0.9 - 9*	*0x1 (Sent)*

4.4.3.3 Message Pool

Others have proposed using a message pool that includes details about the received and sent messages [40]. We agree with the pertinence to use such a structure and propose incorporating the following fields:

- A packet identifier unique to the node. It is not necessarily sent with packets as they travel through the network—this decision has to be made by a protocol.
- An XML tag or an integer describing the data.
- The priority of the packet.
- The number of packets in the burst.
- The status of the packet, *i.e.*, whether it has been successfully sent or not. This field allows protocols in the stack to maintain queues.

We summarize these elements in Table 4.5. Such a structure, combined with the neighbor table, can help several layers make decisions about the routing and sleep schedules or media access for a packet.

4.4.3.4 Accessing the Structures

Read and write access to all structures may only be granted through *CLOI*: this has the combined advantages of atomicity[1] and modularity. Because read and write operations are placed in atomic segments, they are executed in the order they are received, without prioritization. X-Lisa also provides additional functions for access management: because protocols do not know the identity of the neighbors present in the table *a priori*,

---

[1] Whereby the same segment of code may be accessed by only one element at a time

they may invoke the function `nextEntry`, which returns a pointer to the next entry in the neighbor table. A returned $NULL$ pointer indicates that the bottom of the table has been reached.

4.4.4 Event Signaling

X-Lisa provides two classes of event notifications: *protocol events* and CLOI *events*. The former designates events generated by protocols in the stack and relayed directly through *CLOI*. The type of the event is not known to *CLOI* but has to be meaningful to both the provider and user of this event. CLOI *events* refer to a set of events defined in X-Lisa and that include notification of a new packet in the pool, a new neighbor, a full neighbor table, etc. We expect that CLOI *events* will be much more common than *protocol events*[2], and thus a programmer can choose to not use *CLOI* events at compile time if they are not needed.

Protocols that require event signaling can subscribe to *CLOI* at compile time. Because NesC does not allow runtime dynamic wiring, a protocol may not unsubscribe from event notifications. The MeshC [66] language overcomes this limitation. Either way, this does not seem to be a strong constraint as we expect protocols to have an interest in event notification for the duration of the network lifetime. If no longer relevant, event notifications may simply be ignored by subscribing protocols.

4.4.5 Information Exchange

In order to keep the information contained in the neighbor table, X-Lisa provides an automatic update service. The information exchange is carried by an *information vector* that updates the neighbors of a node.

4.4.5.1 Information Vector

The information vector includes some or all of the fields necessary to populate the neighbor table. These fields will automatically be filled by the *CLOI* of the sending node and read by the *CLOI* of the node's neighbors. The information vector may be

---

[2] In our implementation of Section 4.6, one or two *CLOI* events were generated every time a packet was received or sent.

Table 4.6: A packet with a *CLOI* information vector piggy-back (TOS_Msg fields not included).

ID	Content	Vector	data
2B	1B	nB	xB

piggy-backed onto broadcast packets or sent as a stand-alone packet when no broadcast packets are sent for a certain amount of time. The update mode may be set through a *CLOI* function that turns automatic updates and piggy-backs on or off. The inherent mode of propagation for the information vector is one-hop broadcast. This implies that when the information vector is piggy-backed to packets traveling more than one hop away, the contents of the vector change at every hop.

CLOI retains the principle of abstracted encapsulation that guarantees that X-Lisa components need not be informed of other protocols' data structures and packet headers.

One limitation introduced by the encapsulated piggy-back information vector is that it forces every node in the network to run compatible versions of *CLOI*: otherwise, any layer above *CLOI* could not receive correctly formatted packets.

4.4.5.2 Size of the Information Vector

Not all fields may require a frequent update depending on the application QoS requirements, the needs of the protocols in the stack, or the nature of the field itself. *CLOI* piggy-backs only the required parameters to its neighbors. These can be requested through a *CLOI* command for each parameter. *CLOI* fetches fields from the neighbor table and copies them into the piggy-back. The structure of the piggy-back and stand-alone update is illustrated by Table 4.6.

To inform receiving nodes of the content of the information vector, we include a *content* field in *CLOI* messages. This field is one byte long, and each bit designates a parameter in a determined order.

CLOI exchanges information of the fields that are requested by at least one protocol through the `exchangeField` function by invoking:

```
Cloi.exchangeField(uint8_t field, bool comm);
```

where `comm` is `FALSE` when the field is not required by the protocol, `TRUE` otherwise. A `FALSE` command decreases the integer value corresponding to the `field`. When this value is equal to 0, *CLOI* considers there is no need to exchange the parameter.

This Key-Length-Value solution is the first one of its kind for packet exchanges and, together with the structure of the neighbor table, allows changing fields of interest quickly.

4.4.5.3 Frequency of updates

Dedicated services within *CLOI* take care of updating the fields of the neighbor table and message lists. However, some protocols exchange important data about their status at pre-determined times (*e.g.*, GFG [12] and GPSR [13]), and this may not agree with the schedule imposed by *CLOI*. Thus, *CLOI* has an automatic update knob that such protocols may control. At the protocol's request, *CLOI* will automatically perform an update function, sending out a vector with its information.

Very frequent updates may prove unnecessary and bandwidth consuming. Thus, *CLOI* will not append the information vector to a packet succeeding another by less than a predetermined time (t_{update}), in order to reduce the overhead imposed by this architecture. This predetermined time can either be fixed or adaptive depending on whether or not *CLOI* can *learn* about the rate of change of information in the neighbor table.

4.4.6 Maintenance of the Neighbor Table

In order to maintain the freshness of the neighbor table, *CLOI* must detect obsolete information and remove the corresponding entry after a node has died or moved away. Every time data is added to the neighbor table, a time stamp is created or modified corresponding to that information. A timer runs in the background, and every $t_{cleanup}$, *CLOI* checks for entries that have not been updated for a given period of time ($t_{obsolete}$). A protocol may block this process—independently scheduled at every node—to avoid discrepancies in neighboring sensors' tables and directly call for an inspection of outdated entries when it finds convenient.

If the neighbor table reaches near capacity, entries may be removed even if its information is not stale. However, since some neighbors may be more important than others, protocols may signal so by setting the field *hold*, which signals that a neighbor should not be removed if the table is full. Neighbors not *held* will be first choices to be removed.

4.4.7 Important Services

As mentioned previously, one of the advantages of X-Lisa is that it allows protocols to re-focus on their primary functions. To enable this, X-Lisa adds peripheral services that supply some of the information needed to fill *CLOI*'s information repositories. These include the following:

- ID assignment service. This could be as simple as predetermined ID numbers hard-wired in the sensor, or this could be a service-type identifier, also used for routing, as proposed in [49]. Although it was not originally intended for them, X-Lisa may support IP-based networks thanks to additional ID services that maps IP addresses to supported $16\ bit$ addresses.

- Location service. The geographic coordinates of a sensor are indispensable and usually assumed in most research works in the field of sensor networks. However, GPS solutions are costly. In [67], Rao et al. propose a solution that assigns virtual coordinates to each node and allows geographical routing.

- Time synchronization service. Many protocols assume that a common time is precisely shared among all nodes in the network. Several strategies have been proposed, including one based on periodic synchronization packets as in RBS [45], or by estimating the time delay between a sender and a receiver [44].

- Channel estimator. Several strategies are possible, from reading the back-off value for CSMA schemes, to one-hop packet delivery ratios, or measurements of the signal strength coming from a neighbor. A realistic scenario using the MoteIV Tmote Sky [68] sensor nodes is to read the LQI field provided in received packets, and to keep track of the one-hop packet delivery ratios at either the data link level (when ACKs are enabled) or at the transport layer otherwise.

- Remaining energy measurement. Although the battery voltage degrades in a non-linear fashion, this is a good indicator (when mapped through a look-up table) of the remaining energy at the node. A service is needed to access the current battery voltage and perform the mapping from voltage to remaining energy.
- Finally we can imagine more complex services such as a sensing ability service. Nodes may completely or partially lose their sensing abilities over time because of physical degradation or aging. A complete loss of a sensor is fairly easy to detect since no response is to be expected from it. However, assessing that a sensor is providing faulty data is a difficult task, as the sensor will return plausible but incorrect values for a variable. At this point, we believe that such a service would require the cooperation of other sensors and the use of fuzzy logic or neural networks.

These services are available as libraries and thus individual services can be selected during compilation (*i.e.*, only those services that are needed for the particular set of protocols in the stack would be selected). Furthermore, selected services can be turned off during run-time to guarantee the maximum flexibility.

4.5 Implementation Details

We implemented the core features of X-Lisa in TinyOS 1.1.15 for the Tmote Sky platform. We also added DAPR [6] [69], a cross-layer routing and node activation protocol to X-Lisa. In this section, we provide detailed information about our implementation.

4.5.1 Components in TinyOS

The TinyOS code is organized as depicted in Figure 4.3 CloiMessageHandler is the name of the component corresponding to the horizontal layer of *CLOI*. Protocols wishing to send a packet down the stack must wire to the **Send** and **Receive** interfaces of CloiMessageHandler, which takes care of appending the information vector. MHEngineM is a module inspired by P. Lewis et al. [70] and that provides multi-hop services. The existence of this module follows proper TinyOS programming guidelines rather than a requirement for X-Lisa.

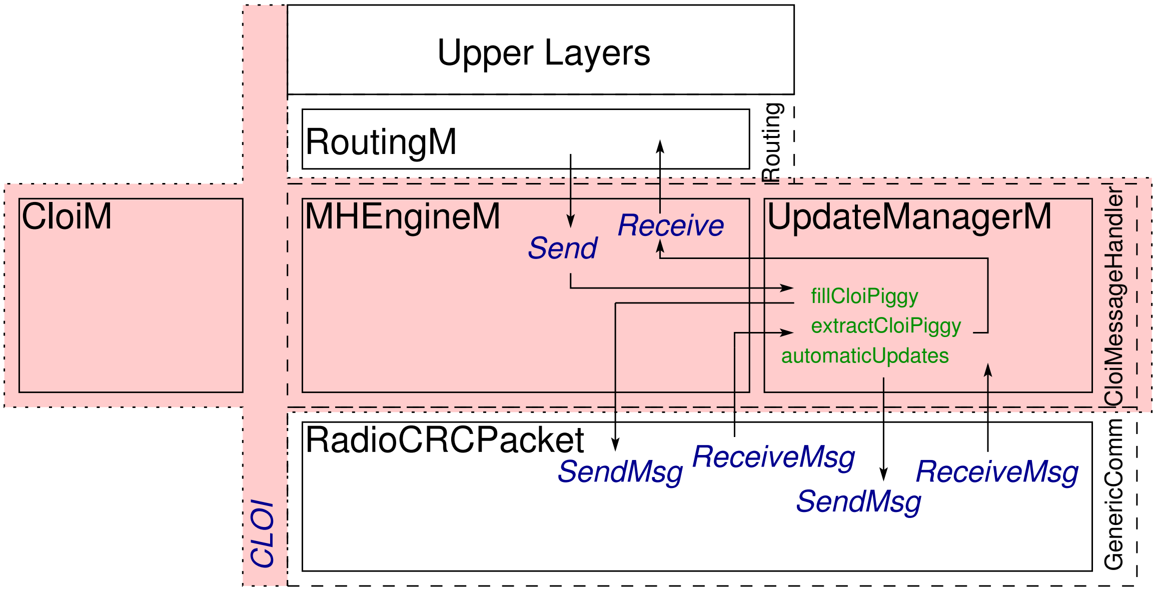


Figure 4.3: The X-Lisa architecture in TinyOS. The shaded area corresponds to *CLOI*. Implementations are in large characters and delimited by solid line frames. Each component is bounded by a dashed line, and its name appears vertically.

To support encapsulation, lower layers should be unaware of packet structures from upper layers. Consequently creating packets with two or more structures of varying size is a difficult task. Packets received from upper layers should be considered as data. Traditional packets may define a "data" field thusly: `uint8_t data[TOSH_DATA_LENGTH` `-` n`]`, where n is the number of bytes taken by the other fields in the packet. In other words, upper layers may dispose of `TOSH_DATA_LENGTH -` n bytes for their needs, although they may not use all of them. The MAC layer then only sends the first bytes in the packet, and discards the unused part in the *data* field. Table 4.7 shows a packet definition example.

CLOI packets define a *data* part as well as the information vector of changing size. Consequently, *CLOI* must define packets with two fields of maximum fixed size. However, the MAC protocol can only cut unused bytes placed *at the end* of a packet. Table 4.8 illustrates this problem.

To prevent such waste of bandwidth, the bytes from the *data* field are moved to the unused sections of the *CLOI* information vector. When receiving a message, *CLOI* performs the inverse operation and transfers bytes from the information vector belonging

Table 4.7: Example of a routing packet definition. The application layer may dispose of the space defined by *data*. The MAC protocol only sends the used bytes (S) in the packet, and discards those that are not in use (X).

int16_t Destination	int8_t TTL	uint8_t *data*[MAX - 3]											
2B	1B	MAX - 3											
S	S	S	S	S	S	S	S	S	S	X	X	X	X

Table 4.8: The MAC protocol only ignores bytes placed at the end of a packet (designated by X). If two fields of varying size are defined, the MAC sends a section filled with unused and unassigned bytes (U).

int16_t node id	uint8_t info. vector						uint8_t *data*[MAX - m -3]						
2B	mB						MAX - m - 2						
S	S	S	S	X	X	X	S	S	S	X	X	X	X

to the *data* field (see Table 4.9). With this simple scheme, *CLOI* sends only the required information without any supplemental overhead.

4.5.2 Information Storage

4.5.2.1 Neighbor Table for X-Lisa

The fields of the neighbor table are:

- Node ID, an unsigned *long* (2 bytes in TinyOS).
- Time stamp, a structure of two 4 byte integers provided by TinyOS expressing the time in milliseconds.
- The table is an array of 1 byte integers that can be filled by specialized functions. Its size can be limited to that of only the parameters needed by the protocols.
- Hold, a boolean.

Table 4.9: *CLOI* rearranges bytes in order to place all unused bytes at the end of a packet. The MAC protocol may then discard the unused bytes.

int16_t node id	uint8_t info. vector						uint8_t *data*[MAX - m -3]						
2B	mB						MAX - m - 2						
S_0	S_1	S_2	S_3	U_4	U_5	U_6	S_7	S_8	S_9	X	X	X	X
Rearrangement													
S_0	S_1	S_2	S_3	S_7	S_8	S_9	X	X	X	X	X	X	X

4.5.2.2 Sink Table and Message Pool

The sink table and message pool elements are represented in the same way as the corresponding elements of the neighbor table. For the message pool, a 1 byte integer describes the data contained in a packet. Another byte gives the packet's priority (most significant bit set to 1 indicates high priority) and the number of packets in the burst (7 bits). A sink table of four sinks requires 44 bytes of memory. A message pool of ten packets requires 40 bytes of memory.

4.5.3 DAPR and the Proposed Architecture

In this section, we illustrate how X-Lisa can be used with existing protocols, namely DAPR and GFG / GPSR.

DAPR is a cross-layer protocol that jointly performs routing and node activation. In randomly deployed heterogeneous networks, sub-areas of the monitored region may suffer from low levels of coverage. Nodes located in scarcely covered areas are critical to the global network coverage and need to be preserved. DAPR recognizes these sensors by assigning them a high application cost, and it avoids routing other nodes' packets through them. The definition of application cost depends on the nature of the application, such as geographical monitoring, as first introduced in [6], or health monitoring [69].

The DAPR protocol is divided into rounds of three phases:

1. Route Discovery Phase: the sink floods a *query* and sets up costs. When a *query*

is received for the first time, a sensor initializes a *query* forward timer that is proportional to its cumulative route cost and an *opt-out* timer that is inversely proportional to its cumulative route cost. This guarantees that when a node receives a first *query*, it is through the lowest-cost path.

2. Role Discovery Phase: when the *opt-out* timer expires, a node checks whether its coverage is needed. If not, it beacons to its neighbors to indicate its intention to sleep in the round. The timers are set so that a sensor with a high application cost will be among the first to opt out.

3. Query Processing Phase: this is the phase of normal operation. Nodes forward packets to the sink through their neighbor presenting the smallest cumulative cost.

Next we will see how each of these functions is performed and optimized within X-Lisa.

4.5.3.1 Route Discovery Phase with X-Lisa

DAPR's routing layer originates the *query* and requests *CLOI* to fill the message pool with a description of the packet (*e.g.*, *network administration*) and the packet's priority (high). The *query* contains the issuing node's value of its current application cost, and *CLOI* appends the information vector to this *query* packet.

The application cost is calculated using values already available in the neighbor table. DAPR sends *CLOI* a read command for the neighboring nodes' information—remaining energy, sensing abilities and monitoring entity. Since DAPR need not keep its own list of neighbors, it utilizes *CLOI*'s *next entry* function. DAPR starts at the node's own *ID*, and loops through the entire list until the function returns a $NULL$ pointer (signifying the end of the table).

The field "application cost" is easily added to the neighbor table thanks to the flexible nature of the structure. Thus the application cost will be made available to the routing and node activation subroutines of DAPR, as well as any other layers that potentially could benefit from this information.

When a *query* is first received, DAPR blocks the *CLOI* interface from updating and cleaning entries during the route and role discovery phases. This prevents two neighboring sensors from having time-differing views of the same area of the network—and consequently from calculating inaccurate application costs and wrongly assessing their usefulness to the network. DAPR forces a cleaning of the outdated entries at the end of the role discovery phase and then restores the automatic update feature of *CLOI*.

DAPR requires time synchronization so that all nodes terminate the DAPR round at approximately the same time. *CLOI* guarantees that the nodes' clocks are synchronized to within an ϵ that is dependent on the particular time synchronization service implemented within *CLOI*. Consequently, the *query* timestamp serves as an anchoring time for the round.

4.5.3.2 Role Discovery Phase with X-Lisa

During the role discovery phase, *opt-out* timers are set. Once the timer expires, if the MAC protocol is not experiencing delays (*e.g.*, it uses a TDMA protocol or dedicated channels for network administration), the usefulness of the sensor can be checked immediately. If the node's sensing abilities are not needed, an *opt-out* packet is sent down the stack with the appropriate accompanying description and priority. On the other hand, if the MAC protocol incurs delays (*e.g.*, IEEE 802.11 in high congestion scenarios), the MAC protocol needs to check the sensor's usefulness immediately before it passes the *opt-out* packet to the physical layer. Whether at the node activation or MAC layer, checking the node's redundancy is a simple process where only read requests are sent to *CLOI*.

When a sensor receives an *opt-out* packet, it is sent up the stack until the node activation protocol requests a write from *CLOI* and sets the source of the *opt-out* packet as being inactive for the rest of the round (*e.g.*, *CLOI* sets the associated neighbor's *status* field in the neighbor table to "inactive").

Algorithms 1 and 2 illustrate the many interactions between DAPR and *CLOI*.

4.5.3.3 MAC Layer with X-Lisa

For the best performance with CSMA schemes, the MAC protocol needs to be modified to check the sensor's usefulness, as well as for making smarter decisions about sending packets. One evident drawback of DAPR is that *queries* may be lost or significantly delayed when sent over a network reporting large amounts of data to the base

```
Require: Received Query, Allocated pointers
  if !first query then
    Cloi.updateField(query→src, &query→cost, COST);
  else
    Calculate Cost:
    nextEntry = Cloi.readEntry(TOS_LOCAL_ADD);
    thisEntry = Cloi.readEntry(TOS_LOCAL_ADD);
    ownEntity = *(uint8_t*)VALUE_ENTRY(thisEntry, ENTITY);
    ownSensing = *(uint8_t*)VALUE_ENTRY(thisEntry, SENSING);
    ownEnergy = *(uint8_t*)VALUE_ENTRY(thisEntry, ENERGY);
    while nextEntry $!= NULL$ do
      entity = *(uint8_t*)VALUE(nextId, ENTITY);
      sensing = *(uint8_t*)VALUE(nextId, SENSING);
      energy = *(uint8_t*)VALUE(nextId, ENERGY);
      if entity $==$ ownEntity then
        $divVector+ = sensing \cdot \frac{1}{energy}$;
      end if
      nextId = Cloi.nextEntry(nextEntry-¿id);
    end while
    cost = $\sum \frac{ownSensing \cdot \frac{1}{ownEnergy}}{divVector}$;
    cumulativeCost = query→cost + cost;
    Cloi.updateField(TOS_LOCAL_ADD, &cumulativeCost, COST);
    set QueryTimer($\alpha \cdot cumulativeCost$);
    set OptOutTimer($\beta \cdot \frac{1}{cumulativeCost}$);
  end if
Require: QueryTimer Expires
  sendQuery(*(float*)VALUE(nextId, COST));
```

algorithm 1: Route Discovery Phase

station. DAPR's current functionalities ensure that nodes stop sending data packets to the sink during route and role discovery phases. Consequently, the network is only semi-functioning during these two phases: packet delivery is not guaranteed, nor does it use optimal paths.

```
Require: Received OptOut, Allocated pointers
    bool fasleValue = OFF;
    Cloi.updateField(OptOut→src, &falseValue, STATUS);
Require: OptOutTimer Expires
    Evaluate Usefulness of Node:
    nextId = Cloi.readEntry(TOS_LOCAL_ADD);
    thisEntry = Cloi.readEntry(TOS_LOCAL_ADD);
    ownEntity = *(uint8_t*)VALUE_ENTRY(thisEntry, ENTITY);
    ownSensing = *(uint8_t*)VALUE_ENTRY(thisEntry, SENSING);
    while nextEntry $!= NULL$ do
        entity = *(uint8_t*)VALUE(nextId, ENTITY);
        sensing = *(uint8_t*)VALUE(nextId, SENSING);
        status = *(uint8_t*)VALUE(nextId, STATUS);
        if status AND (entity $==$ ownEntity) then
            coverage $|=$ sensing;
        end if
        nextEntry = Cloi.nextEntry(nextEntry-¿id);
    end while
    if (coverage $|$ ownSensing) $\leq$ coverage then
        sendOptOut();
    end if
```

algorithm 2: Role Discovery Phase

X-Lisa allows for some changes to the way DAPR operates. *Queries* and *opt-out* packets have high priorities and are specifically described in the message pool. The new MAC protocol can decide to delay, or even drop, data packets when it witnesses high congestion down the path to the sink, and choose to favor network administration packets.

4.5.3.4 Physical Layer with X-Lisa

Finally, the physical layer does not need to concern itself with keeping track of the sensor's location and that of its neighbors, nor do any of the upper layers need to include the distance to the next-hop in the packet header. The physical layer can retrieve such information through the neighbor table using the packet's next-hop field as the node ID in the read request to *CLOI*.

4.5.4 GFG / GPSR and X-Lisa

GFG and GPSR [12] [13] are geographical routing protocols that extensively use the nodes' location to select the next hop that sends packets closer to the destination. The sensor's location is exchanged between neighbors using *beacons*. The packet delivery ratio strongly depends on the *beacon* interval: when coordinates are updated frequently, the packet delivery ratio increases, although the protocol incurs more overhead. GFG / GPSR may find that the exchange of information as done by *CLOI* is not appropriate. GFG / GPSR may thus turn off the automatic update service within *CLOI* and send periodic *beacons* at intervals they deem appropriate. To do this, GFG / GPSR broadcasts an empty packet; *CLOI* then piggy-backs the information vector, taking care of the actual propagation of the coordinates.

Otherwise, a location service in X-Lisa periodically updates the node's coordinates and *CLOI* exchanges this information with the nodes neighbors. Thus the neighbor table is kept up-to-date on the location of the node and each of its neighbors, removing the burden of performing this location service from the routing protocol. Algorithm 3 describes how a geographical routing protocol may take advantage of some functions in X-Lisa.

However, strictly location-based routing may not always be efficient. A sensor geographically close to the destination does not necessarily enjoy a high bit rate or low delay. We contend that modifying the protocol stack is made easy by X-Lisa. *CLOI* contains information about the link quality from a node to each of its neighbors, which can be directly exploited by the routing or—less probably—MAC protocols.

4.6 Results

In this section, we intend to show the benefits of using X-Lisa through simulation of an existing protocol whose behavior with and without X-Lisa was studied in TOSSIM, the TinyOS simulator. The choice of running a simulation rather than collecting data

```
    Cloi.setAutomaticCleanUp(FALSE)
Require: get Location
    Cloi.updateField(Location→src, &Location→X, XLOC);
    Cloi.updateField(Location→src, &Location→Y, YLOC);
Require: Forward Packet p, Allocated pointers
    sinkId = p→dest;
    sinkEntry = Cloi.readSink(sinkId);
    sinkX = sinkEntry.xLoc;
    sinkY = sinkEntry.yLoc;
    nextId = nextHop = TOS_LOCAL_ADD;
    nextEntry = call Cloi.nextEntry(nextID);
    dist = $\infty$;
    while nextEntry $! = NULL$ do
        X = *(uint16_t*)VALUE_ENTRY(nextEntry, XLOC);
        Y = *(uint16_t*)VALUE_ENTRY(nextEntry, YLOC);
        status = *(uint8_t*)VALUE_ENTRY(nextEntry, STATUS);
        if status then
            dist = min(dist, computeDist(X, Y, sinkX, sinkY, &nextHop));
        end if
        nextEntry = Cloi.nextEntry(nextEntry-¿id);
    end while
    if nextHop $! =$ TOS_LOCAL_ADD then
        p→addr = nextHop;
        send(p);
    end if
```

algorithm 3: GFG and *CLOI*

from an actual deployment had two motivations. First, it is easier to collect data with TOSSIM than through a sometimes faulty gateway to Matlab. Our objectives in this section offer a second reason: we are trying to isolate the benefits of using X-Lisa, and would like to factor out the unpredictabilities of the data link and physical layers.

4.6.1 Modus Operandi

4.6.1.1 Qualitative Study

Ease of use and maintenance, simplicity, and generality of an architecture are not easily quantifiable, and are left for the most part to the appreciation of every programmer. However, we begin by providing a data-point that illustrates the simplicity and generality of X-Lisa through protocol swapping. We demonstrate the power of our information-sharing architecture through the comparison of two protocol suites.

Our starting point was XLM [43], which exhibits total layer fusion and is thus the counter-point of X-Lisa. We wanted to see if X-Lisa was rich enough to replicate the behavior of XLM while maintaining the convenience of separated protocol layers. We illustrate the advantage of keeping a layered scheme by swapping the MAC protocol from the original XLM MAC functions to a Low-Power-Listening (*LPL*) scheme [22].

4.6.1.2 Quantitative Study

We then quantify some of the limitations and gains of using X-Lisa such as the extra-overhead and increase in quality of service (QoS) induced by X-Lisa. There exists a plethora of protocols for WSN, many of which could benefit from X-Lisa. We selected DAPR [69], a distributed fused-layer routing and node activation protocol, for our familiarity with the protocol and because it strikes a good balance between cross-layer (combined routing and node activation) and layered (interface of DAPR with other protocols in the stack) schemes.

We implemented the original version of DAPR as well as a modified version of DAPR that takes advantage of the X-Lisa architecture. In our simulations, we measure the total number of unicast and broadcast packets, which provides an indication of the extra energy required by X-Lisa. We also quantify the packet delivery ratio, as well as the number of reports delivered to the data sink: together, these can measure the QoS provided to the end application.

Since the goal of this section is not to evaluate DAPR, we limit the simulation to a simple scenario: a likely candidate to route other nodes' packets is mobile and may move around (on average, every $150\ s$). We use a small number of nodes (5 and 10) so that we may easily interpret the behavior of the network. Had we chosen several mobile nodes, we would have tested the resilience of DAPR, rather than the benefits brought by X-Lisa.

4.6.2 Qualitative Study: The Expressiveness of X-Lisa

In this section, we show that X-Lisa is an architecture that is rich enough to mimic the behavior of XLM, a fully-fused scheme. XLM is particularly of interest because it represents an extremum in cross-layer designs, and hence is a candidate of choice to test the limits of X-Lisa.

4.6.2.1 Implementation Details

XLM establishes unicast communications through an RTS / CTS handshake before DATA is exchanged and acknowledged. Power is saved through a duty-cycle, which turns the radio chip on and off periodically: nodes remain asleep for the remainder of the cycle if they cannot participate in a communication ($\mathcal{I} = 0$) or if they lose contention to another node. Contention is handled through a CTS response backoff proportional to the node's distance to the destination. The stateless greedy routing is receiver-based and the node sending the first CTS signals its intention to forward a packet. Congestion control reduces the application packet generation rate in case of communication failure, and increases it otherwise.

In the initial protocol suite, we decomposed XLM into the five layers (+application) of Figure 4.2 and included them to the X-Lisa architecture. The new entity called XLM / X-Lisa is the layered version of XLM. The transport layer now extracts information from user data (and updates some fields in X-Lisa's neighbor table) and segments long data packets. In this suite, the network layer has a limited role, and keeps a queue of packets. The link layer performs the RTS / CTS / DATA / ACK hand-shake and controls the radio chip for duty-cycle. Among other things, X-Lisa shares packet delivery failure information to the application layer for congestion control purposes and uses the neighbor table as a central storage of node positions for location look up. Figure 4.4 shows the new organization.

We tested XLM / X-Lisa and found that it replicates the behavior of XLM while retaining a layered structure (routing packets to the destination using the same number of packets).

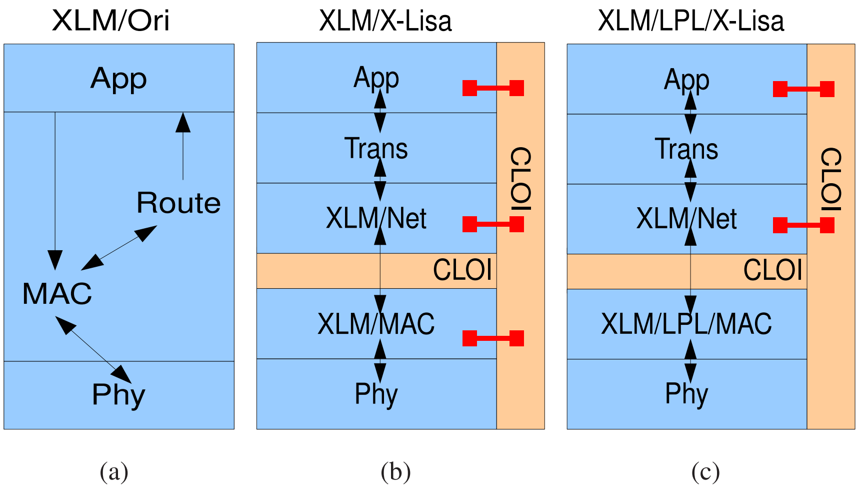


Figure 4.4: The original XLM (a), was broken into a layered scheme (XLM / X-Lisa) (b), and its MAC layer was replaced (c). Arrows show packet exchanges between layers, and squares information exchange.

4.6.2.2 Protocol Swapping

During the introduction of this chapter, we conjectured that protocol maintenance would be eased by modularity. To test this supposition, we attempted to swap MAC protocols to a *LPL* MAC protocol. Although we cannot quantify the ease with which we did so, a successful MAC replacement is meaningful in itself.

The second protocol suite is a variant of XLM / X-Lisa: the original MAC layer was replaced by the *LPL* MAC protocol SpeckMAC-D [22] as illustrated in Figure 4.4. We named the new entity XLM / LPL / X-Lisa. In SpeckMAC-D, every node sleeps for t_i s (the inter-listening time) between wake-ups. In order to guarantee that the receiver will wake up at some point during a transmission, a sender must repeat the same packet for t_i s. If a node awakes and receives a packet, its MAC protocol forwards it to the network layer before sleeping for the rest of the cycle.

Similarly to XLM, we modified XLM / LPL / X-Lisa to route packets only if the node is the closest to the destination, a strategy akin to restricted flooding.

4.6.2.3 Successful Swapping

The two suites of protocols were implemented in TinyOS and simulated with TOSSIM. We conducted simulations on 10 nodes with the topology of Figure 4.5. The Source

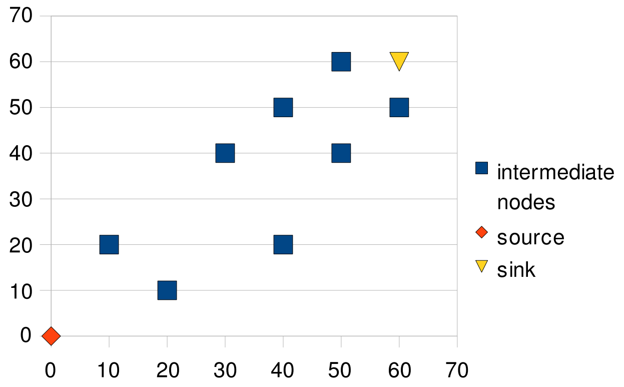


Figure 4.5: Simulated network topology (units in meters). The nominal radio range is $30\ m$.

	XLM/LPL/X-LISA	XLM/X-LISA
Received Packets	165	2036
Sent Packets	3674	378
Goodput (%)	0.75	1.0
Latency (s)	1.503	0.683

Table 4.10: Selected metrics comparing the behaviors of XLM / X-LISA and XLM / LPL / X-LISA.

node $(0,0)$ sends a packet to the sink node $(60,60)$ every $5\ s$, for a total simulation time of $100\ s$. The results are shown in Table 4.10.

The number of sent and received packets should be seen as a rough indicator of the energy consumed by each suite, with certain caveats. A CSMA-based MAC protocol forces the radio to stay in idle mode (a state that incurs the same energy consumption as active receiving mode) significantly more than a *LPL* scheme. Consequently, sending more packets with a *LPL* MAC protocol does not always result in increased energy consumption.

Results show that XLM / LPL / X-LISA sends more packets than its original counterpart because the *LPL* scheme repeatedly sends packets over $t_i\ s$. The RTS / CTS handshake accounts for over a third of the 378 packets counted for the XLM / X-Lisa suite. On the other hand, XLM / X-LISA receives more packets because every communication requires a hand-shake, and because many nodes receive RTS / CTS / DATA / ACK packets even though they lost the contention to other nodes and are not part of the communication. Finally, both suites exhibit high goodput (greater than 75%), with XLM / X-Lisa showing the better performance. Likewise, XLM / X-Lisa yields lower latency. Both metric differences can be explained by the fact that nodes that are candidates to participate in a communication ($\mathcal{I} = 1$) are always on during a cycle, allowing few packet to be dropped or delayed.

These results show that the replacement of the original XLM MAC protocol by SpeckMAC-D led to observably similar behaviors: according to a set of quantifiable metrics, the two suites are within the same order of magnitude. Thus, swapping protocols was possible and X-Lisa did not degrade the performance of the protocol. What these results do *not* demonstrate is that one particular MAC protocol yields a longer lifetime or any other desirable QoS improvements over the other, because a protocol must take advantage of the information brought by X-Lisa.

4.6.3 Quantitative Study: Measurable Cross-Layer Improvements

We now show that X-Lisa helps improve the performance of the network by allowing cross-layer interactions to their fullest.

4.6.3.1 The DAPR Protocol

DAPR assigns "application costs" to all nodes by periodically flooding a *query* request to the network: the higher the cost, the more important the node is to the application (because it may be fitted with unique sensors or because it is located in a sparsely covered area of the network, etc.) Nodes with high costs are eager to deactivate and make poor choices as relays for other nodes' packets. For the purpose of this simulation, we divided the whole network into *zones*: nodes in the same zone may communicate with one another, and with nodes from adjacent zones. A routing tree can thus be formed, and nodes with low costs route data packets to a single data sink. Nodes located in a target zone repeatedly send data reports at a rate of $0.2pkt.s^{-1}$.

Because DAPR sends queries at the beginning of every $60\ s$ round, changes happening to the tree are unknown to the protocol stack until a new query is flooded, with sometimes serious consequences. For instance, if a relay node moves from one zone to a neighboring zone, data packets will stop being delivered.

X-Lisa brings new information to the protocols it serves: for the case at hand, DAPR can now be notified when a change of zone occurs or if a new neighbor has been added to its table. Such changes usually mean that a node's next-hop neighbor might have changed, and that it may need to start or stop sending data reports. One important metric is the average update delay, defined as the time between a change and its notification to the nodes' neighbors. The longer the delay, the longer nodes affected by a change fail to take appropriate corrective measures.

Because it is able to act only when a change is detected, *i.e.,* prior to a tree breaking, DAPR does not require automatic updates, although they may be useful to other protocols. Conversely, without the added knowledge provided by X-Lisa, DAPR must include extra information (such as a node's current zone) along with queries, regardless of whether a change has indeed happened.

4.6.3.2 Simulation Results

Since the goal of this section is not to evaluate DAPR, we limit the simulation to a simple scenario: a likely candidate to route other nodes' packets is mobile and may move around (on aver, every $150\ s$). We use a small number of nodes (5 and 10) so that we may easily interpret the behavior of the network. Had we chosen several mobile nodes, we would have tested the resilience of DAPR, rather than the benefits brought by X-Lisa.

We implemented DAPR with and without X-Lisa in TinyOS. While we ran simulations in TOSSIM, we also compiled the code for the Tmote Sky platform. DAPR alone takes up approximately $21.5\ KB$ of ROM, and $1.3\ KB$ of RAM. With X-Lisa (and all its features), these numbers become $34\ KB$ and $1.9\ KB$, which can easily be accommodated by the Tmote Sky.

Figure 4.6.3.2 presents the number of unicast and broadcast packets for DAPR alone and DAPR with X-Lisa. Also shown are the number of report packets that were sent and the average packet size. For both 5 and 10 nodes, the number of unicast packets increases when X-Lisa is used. Since updates are broadcast packets only, this does not constitute overhead for X-Lisa, but merely increased data traffic brought about by im-

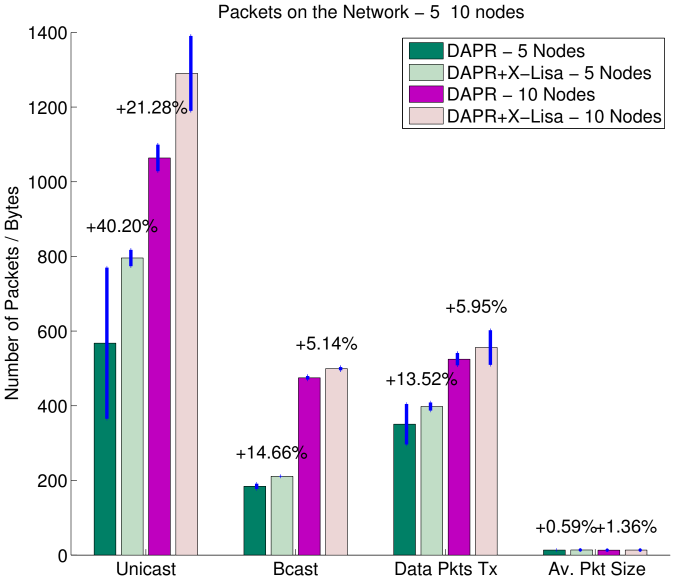


Figure 4.6: Comparison of the impact of DAPR and DAPR + X-Lisa on the network for 5 and 10 nodes. The numbers on the graph are the relative change from DAPR to DAPR + X-Lisa.

proved performance of DAPR with X-Lisa. The increase in broadcast packets remains modest (a maximum of 15%) even for 10 nodes and measures the added cost of X-Lisa. The number of data reports sent to the data sink differs only slightly: this is because when a relay node moves to and from the target zone, X-Lisa can notify its application layer to start or stop sending reports. But in the absence of X-Lisa, the application layer does not learn of it until the next DAPR round, which causes the node to send too many or too few data reports. Over the whole node lifetime however, these tend to average out. Finally, the average packet size stays approximately the same: the information vector sent by X-Lisa does not need to be exchanged if no movement between zones has been recorded. This compensates for larger packets when the full information vector is present in all packets.

Figure 4.7 measures the advantages provided by X-Lisa: a net decrease in the average update delay, which translates into significant gains of QoS. Neighbors of a moving node are notified of a change up to 85% faster when X-Lisa is used. Protocols may take advantage of this in several ways, with varying effects on different metrics. With our implementation of DAPR, this translates into a steep increase in the number of packets

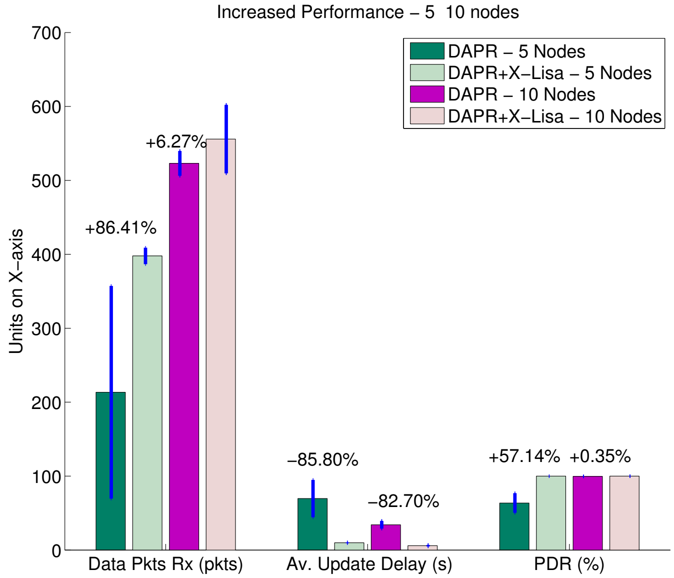


Figure 4.7: Comparison of the QoS of DAPR and DAPR + X-Lisa on the network for 5 and 10 nodes. The numbers on the graph are the relative change from DAPR to DAPR + X-Lisa.

delivered to the data sink, as well as an increase in packet delivery ratio. The magnitude of this increase depends on conditions in the network (when a change happens, how many alternate routes there are, etc.) and on the protocols in the stack; however, the improvement is significant.

These results show that while X-Lisa provides more flexibility and generality, it does not degrade protocol performance. Better yet, correct use of extra information helps increase the QoS to the application, with X-Lisa incurring only small overhead.

4.7 Discussion

In this section, we provide insights as to what makes X-Lisa a stronger architecture for WSN compared with the existing approaches described previously. When selecting a protocol architecture, it is important to consider the specific network for which the architecture is intended. The factors to consider include: overhead imposed by the architecture, generality of the architecture, ease of design and maintenance of the protocols when using the architecture, and the advantages the architecture provides to the application. We examine each of these features.

4.7.1 X-Lisa Extends the Desirable Properties of Existing Architectures

X-Lisa provides the highest level of flexibility available in existing architectures, but it also ensures data freshness and has many features that make it simple to use.

4.7.1.1 Flexibility

The neighbor table within X-Lisa is the most flexible to date for a programming language as simple as NesC: its fields can easily be modified as the needs of protocols change. Swapping protocols is also eased by the layered structure and the *CLOI* interface to the information repositories of X-Lisa.

Moreover, the signaling mechanisms in X-Lisa allow support for many protocols that would otherwise require violations of the layered structure. Only protocols with an interest in event notification have to wire to a simple component. Other protocols do not see such notifications, and thus do not overwhelm the sensor node's compute resources.

Finally, the location of the horizontal layer of *CLOI* renders X-Lisa compatible with SNA if greater platform universality is required. Abstraction of lower layers is guaranteed by the neighbor table and the message pool, as in SNA.

4.7.1.2 Freshness of Information Repositories

Information contained in the tables and message pool is made available to all the protocols as soon as it is updated. Every layer benefits from a fresh view of the node and its immediate neighbors. Automatic clean up services remove stale entries from the neighbor table and prevent it from exceeding its capacity. Moreover, X-Lisa can be ordered to automatically propagate the node information to its neighbors. It does so with very little overhead (typically $1\ B$ for the *content* field).

X-Lisa provides proactive services throughout the architecture. These are self-maintained, and while they can be turned off when they are not required, protocols need not be concerned with the detailed execution of the services. This design choice was prompted by the fact that the various protocols in the stack should not have to perform services such as localization; rather, they should only be responsible for performing their designated task (*e.g.*, routing), for which they should have full control.

4.7.1.3 Simplicity

From its layered structure to its implementation in TinyOS, X-Lisa is the simplest architecture to date. Using X-Lisa requires wiring to only one or two TinyOS components.

Most importantly, and as opposed to proactive services, X-Lisa is not protocol proactive. *CLOI*—which is only an interface—lets the protocols in the stack make decisions in accordance with the information contained in the tables. Cross-layer improvements are only obtained if the various protocols in the stack are able to take advantage of the information provided by the data structures. This ensures that *CLOI* will not perform operations out of the control of the protocols. As a consequence, building a protocol stack with X-Lisa is a simple process that does not require an understanding of the inner workings of *CLOI*. A programmer only needs to learn the input and output functions, conveniently described in the TinyOS *CLOI* interface.

4.7.2 To Use or Not to Use X-Lisa

X-Lisa (and the other surveyed architectures) is not suitable for all sensor network protocol implementations. For example, when the protocols in the stack are very simple or let the base station (with much greater resources) make all decisions about routing and node schedules, as in CentRoute [71], X-Lisa would add too much extra overhead in terms of memory and packet transmissions (see Section 4.6) for no gain to the protocols. In such cases, the programmer should bypass X-Lisa—and many other TinyOS components. However, these cases are uncommon and require specific base stations and node deployments.

On the other hand, many current protocols for sensor networks are designed to take advantage of the type of information that *CLOI* manages. For example, GFG [12]/GPSR [13] require location information. PEAS [9] requires node activation information. Otal et al.'s proposed scheme in [72] uses channel quality estimation and fuzzy logic to select the appropriate next-hop destination at the right data rate. In [73], the authors describe a TDMA based MAC protocol that uses remaining energy information. In these and other protocols, the overhead of obtaining and maintaining the information managed by *CLOI* is unavoidable. In existing protocol stacks, this information must be obtained and maintained by the individual protocols, oftentimes being replicated in multiple layers. The overhead cost for this replicated information is higher than the cost of maintaining the information external to the individual protocols, as is done in X-Lisa. Using *CLOI* to obtain and manage this information removes this burden from the protocols and enables them to focus on their primary tasks.

4.7.3 The Size of X-Lisa

The RAM memory needs are closely dependant on the number of neighbors one node might expect to have. However, this is a problem that any individual protocol using a neighbor table must face. X-Lisa enables the use of a single neighbor table for all protocols, rather than requiring each individual protocol in the stack to maintain its own neighbor table if one is needed. Thus, X-Lisa allows for significant RAM memory savings by eliminating redundant information at various layers. Moreover, the trend in sensor network motes has been geared toward providing smaller platforms with increases in RAM and ROM capacities. Thus, we do not believe that X-Lisa represents a prohibitive cost in terms of memory requirements compared with existing protocol architectures.

In our implementation, the TinyOS X-Lisa component has under 900 lines of code. The EEPROM memory space required for the implementation of X-Lisa is $6\ KB$ without any service libraries (which can be selectively loaded) but with all the TinyOS components necessary for its execution (timers, leds, etc.).

In section 4.4.5.1, we described a mechanism that limits the packet overhead induced by X-Lisa. If all the protocols in the stack have no use for some services or parameters, the information vector is limited to only one byte. Moreover, exchanged information does not always constitute additional overhead—if X-Lisa is not used, this information is just disseminated directly by the protocols.

4.7.4 The Most Gain for the Cost

We believe that there exists a point of optimal benefit for the cost expended in exchanging information among different layers of the protocol stack. Obviously, this depends on the application and the protocols and cannot be found *a-priori* in general. However, X-Lisa was designed to minimize the memory, energy and bandwidth required to propagate node and network information to enable both vertical (within a node's stack) and horizontal (among different network nodes) cross-layer optimizations. Thus we believe that X-Lisa will benefit a large number of existing sensor network protocols. Even more important, we believe that X-Lisa will make it much easier for future protocol designers and system developers to design, implement and optimize their protocols' performance by using the readily available information stores managed by *CLOI*.

4.8 Summary

The improvements obtained through the cross-layering of protocols stems from the information shared among them. Until recently, programmers had to resort to *layer fusion* or to intricate and disorderly designs that curtailed flexibility.

We surveyed the state of the art of information-sharing architectures whose merits include support cross-layer interactions while exhibiting high modularity. We compared these various architectures and found that none of them provide all the requirements needed for sensor networks: support for 1. cross-layer protocols using a modular architecture, 2. services, 3. information exchange suited to sensor network traffic models, and 4. event notification.

Thus, we proposed X-Lisa, an information-sharing architecture that facilitates vertical and horizontal cross-layer optimizations in WSN through a cross-layer optimization interface (*CLOI*). *CLOI* maintains update information on the network state, the nodes' states, the data sinks and the messages to be sent. All layers have access to the information maintained by *CLOI*, which ensures that all protocols in the stack can benefit from cross-layer optimizations facilitated through information-sharing. X-Lisa also prevents use of duplicate neighbor tables at different layers and lets protocols refocus on their core purpose.

In our Tmote Sky platform implementation, we have verified that existing protocols (such as XLM and DAPR) can fit into this information-sharing architecture. However, the real advantage of this architecture is that it will facilitate the design of future protocols by removing the burden of finding, maintaining and sharing important node and network information from the protocols and placing this task within *CLOI*. In spite of some limitations, the ease of use and the implementation freedom of X-Lisa make it a viable option for future sensor network deployments.

X-Lisa standardizes the design of cross-layer protocols by providing a flexible set of parameters to all the protocols in the stack. X-Lisa insures that the information fed to its client protocols is updated and propagated to its neighbors.

However, this part of our work does not consider how to extend middleware interactions to the whole protocol stack, and how to organize the bidirectional flow of information in a manner that is consistent with the goals set in Section 4.3. In the following chapter, we propose an elegant solution to support proactive query notification and manage information between middleware and protocols and services in the stack.

Chapter 5

Supporting Proactive Application Event Notification to Improve Sensor Network Performance

WSNs provide many great challenges because of their resource constraints. Software solutions must operate on a variety of platforms and deployments while providing continuous Quality of Service (QoS) to the end user. In general, QoS should not exceed the level required by the application, as this usually results in depleting network resources faster. Fulfilling this requirement may be further complicated by the inherently dynamic topology of the network, whether sensor nodes are mobile, and whether their energy sources can be replenished.

To support an application, protocols may take advantage of network information, such as knowledge of the resources or locations of the individual nodes. We argued in Chapter 4 that all protocols could benefit from such additional information, but that this wealth of new information requires proper channels for dissemination to a node's protocols and neighbors. For example, ad-hoc violations of an OSI model-based architecture could lead to a "spaghetti design" cautioned against by Kawadia et al. [27]. On the other hand, allowing information-sharing among protocols, as we proposed in X-Lisa (Cross-Layer Information Sharing Architecture), safeguards against this risk while still providing the required information to all protocols in the stack. In particular, X-Lisa shares data repositories among all layers in the stack from the node activation layer to the Data Link / MAC layer through a common interface called *CLOI* (Cross-Layer Optimization Interface). However, X-Lisa does not provide any interface between the application and the protocols in the stack—this is typically the role of middleware.

In [51], Römer et al. define middleware through its purpose "to support the development, maintenance, deployment, and execution of sensing-based applications." To achieve this goal, middleware should necessarily abstract the network mechanisms and heterogeneity. In this chapter, we introduce ideas for information-sharing architectures to support middleware, and we redefine the functions that the application must perform. We also introduce the counter-part of middleware support: protocols in the stack can benefit from information that an application event has occurred. Thus, while the immediate focus of existing middleware techniques is resource, data, or code management horizontally among nodes (*e.g.,* interactions between middleware of two neighbors), we present the idea of vertical integration of middleware (inside a node itself).

We start with the idea that protocols in the stack may benefit from application-related information. For instance, consider a network in which an event (a fire or a structural collapse) is detected and will likely cause node failures. The routing protocol would be better advised to circumvent the event in order to reach the data sink. With this in mind, we propose a *Middleware Interpreter* that logs complex queries from the application. Thus the Middleware Interpreter can determine that an event has occurred. If so, the Middleware Interpreter notifies all subscribing protocols and (if chosen so by the middleware) the node's neighbors, thus shifting the burden of event detection, notification and management away from middleware, which can then revert to its core tasks of data aggregation [74], caching data [54], etc. If middleware chooses to notify direct neighbors about the event, they receive complete information about the query, thus helping in the detection of very complex events and in data publication. These new features entail tight cooperation between the Middleware Interpreter and the neighbor information, as may be stored in a neighbor table, such as that provided in X-Lisa.

Because protocols are regularly introduced or improved upon, the state of the art in WSN protocols is a collection of recent and still evolving contributions. Protocol swapping must be eased so as to promote immediate use of pervasive computing technologies. Therefore, modularity is a key goal of a WSN architecture, although it conflicts with the need for cross-layer interactions. As a consequence, the proposed solution does not allow direct interaction between the Middleware Interpreter and the lower layer protocols. We argue that direct communication between protocols causes “spaghetti” designs and should be avoided. Instead, appropriate interfaces should be used to support information-sharing between the Middleware Interpreter and the protocols.

Using these principles as a guideline, we implement a Middleware Interpreter within the X-Lisa protocol architecture. Using the same *CLOI* interface as in Chapter 4, our proposed solution enables the protocols to interface with the Middleware Interpreter to learn about important application events and adjust their performance accordingly.

5.1 Goals and Challenges of WSN Architectures

In our effort to facilitate the flow of information between middleware and the protocol stack, we propose new guidelines and solutions that should abide by the goals defined for a WSN architecture in Chapter 4. As such, the objectives of flexibility (through increased modularity and universality, service support and event notification), information freshness, low overhead and simplicity remain guiding principles to manage the interactions between middleware and protocols.

To help support these four goals further, we argue that WSN architectures should support middleware. Since the purpose of wireless sensor networks is to serve an application, information about the application is every bit as important to the functioning of the network as local or global network information. Support for middleware, which maps application requirements to the behavior of the network, appears as a key feature of WSN architectures. Application events such as the detection of a fire or the injection of a new query in the network often spur a change in direction for the protocols: respectively, the search of a newer and safer route, or the activation of more nodes.

However, few current middleware solutions share their information with the rest of the protocol stack. This forces middleware to contact protocols individually or to post information that has to be periodically read by various protocols (mobilizing compute resources frequently). Both solutions introduce more violations to layered models (a hindrance to modularity). On the other side of this issue, protocols that do not receive notifications from middleware are left in a reactive position, waiting for events to be brought to their attention (through a specific packet type to send or receive, periodic enquiry of data, explicit signaling from another protocol, etc.) before they can act. We argue that a proactive stance would increase the QoS, which we will show in Section 5.3.

Among resource management middleware [52], only MiLAN [53] suggests direct interactions with protocols in the stack, although in a very ad-hoc manner. This illustrates the need to facilitate the various middleware tasks, which Römer identifies [51] as formulating and dispatching complex sensing tasks and reporting related results. We propose to shift the burden of interacting with the protocol stack away from middleware. While middleware must still supply information understandable by the network, it is how this information is communicated to the stack that is the focus of this work. For instance, while middleware maps application requirements to a set of query conditions, automatic notification of application events by the architecture can simplify the interactions between middleware and the protocols. This allows middleware to concentrate on tasks such as aggregation and other network abstractions.

5.2 Middleware Support

5.2.1 General Ideas

Keeping in mind the goals set for a WSN architecture, we propose to add middleware support. This section does not provide new middleware strategies, but sets to improve communication between existing middleware and other protocols, based on the assumption that the protocols, too, could benefit from application information.

The guiding idea of this work is that protocols should be notified when an application event happens so that they may adjust their behavior to new application and network conditions in a proactive manner. Event signaling is the method of choice so that protocols need not constantly check whether an event has occurred.

In effect, the basic implementation principles are to keep an information-sharing structure with common data repositories storing neighbor, message, and query information. The queries must be stored in a way that is understandable by all the protocols. Since updates of monitored fields are not always meaningful to all the protocols in the stack, only a subset of the protocols can subscribe to information changes. These updates, including those marking the occurrence of an application event, should be signaled through a simple and common interface (in the case of X-Lisa, *CLOI*). To the best of the architecture's capabilities, new information should benefit all protocols and services in a node so as to guarantee the most up-to-date view of the local network and the application. As a new query event fires, subscribing protocols are automatically notified so they may anticipate the new conditions in the network.

5.2.2 Integration Into an Information-Sharing Architecture

5.2.2.1 Motivation and Modifications

In deployments for which the middleware layer is absent from the protocol stack, the application acts as the source of data packets in the network, reacting to a stimulus, or simply following a schedule, usually fixed before compile time. This imposes limitations on the complexity of the task that can be performed. For example, consider the difference between requesting the sensor reading at a specific node (as can be done when middleware is absent) and asking for locations where the sensor reading exceeds a certain value (which requires middleware to manage). The latter case is more of value to data-centric networks such as WSNs and can only be performed with the help of a middleware layer.

Middleware was introduced to map complex application requirements to an abstracted network. These requirements can be expressed by semantically rich queries, whether in SQL-style syntax [74] [54], or not [75]. We assume that the data sink is either able to map end user requests into a query that can be sent to and interpreted by the nodes in the network or that the data sink receives this query already formatted for the network from another party. The query is propagated throughout the network, and it is interpreted and stored by the nodes' *Middleware Solution* (the logical layer in charge of query dissemination, interpretation, aggregation, etc.) and passed up to the application.

For example, the Middleware Solution can receive commands from the end user asking for data reports if a certain sensor reading is below or above a threshold. The Middleware Solution makes the decision to record the query depending on local conditions and parameters in the network. Madden et al. [74] give a good description of

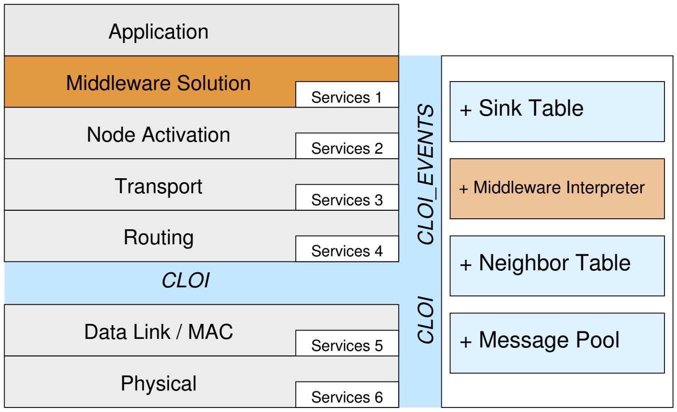


Figure 5.1: New X-Lisa architecture with middleware support.

possible fields in such a query.

However, if the query information is stored within the Middleware Solution only, it may not benefit all layers in the stack. Therefore, we propose a *Middleware Interpreter*, or *MI*. This Middleware Interpreter's main function is to store queries in a form commonly understood by all the protocols. It also makes sense to store queries (that express an interest in a field such as sensor data) closely tied to the structure that manages these fields, the neighbor table. This can be easily accomplished using X-Lisa. A diagram of the updated X-Lisa architecture is shown in Figure 5.1.

5.2.2.2 Query Structure

A query stored in the *MI* is named by a number that helps identify queries uniquely in the protocol stack and throughout the network. A query ID identifies a particular semantic meaning, for instance "high stress level" (when a patient's heart rate and blood pressure are both high). A query is also associated with a field of interest, a date to live ("dtl", the time at which the query expires), and an epoch (the time separating two data reports about the monitored fields). These query fields have to be understood by all the protocols in the stack (this is one of the limitations of this work, although these fields are common in many middleware solutions).

We also added several additional fields to the stored queries in the *MI*:

- A `status` field to signal whether an event is happening ("FIRED") or not ("IDLE"),

Table 5.1: Some of the fields of the Middleware Interpreter with their TinyOS primitive type and an example.

ID	publish	field	dtl	epoch	conditions	composite	status	extra
int8	int8	uint8	tos_time	uint16	uint32	int16	uint8	int8*
6	0	TEMP	0	5000	> 5	-1	IDLE	NULL

- A `publish` field to request that X-Lisa automatically notify its direct neighbors that a query has fired, thus easing collaboration between nodes,
- A `composite` field, because queries may be aggregate (*e.g.,*"send data reports every t_{epoch} s from locations in the network where the temperature exceeds $theta$ and the infrared measurement exceeds φ"),
- A `conditions` fields indicating values under and above which a query event has happened, although this could be replaced by a function wherever simple "higher" or "lower" conditions are not sufficient, and
- Additional memory space (the size of which must be indicated when the query is first added) that can be allocated for the needs of the Middleware Solution or other protocols (*e.g.,* for data caching, node scheduling, event confidence determinations). This is indicated as the field "extra" in Table 5.1.

Table 5.1 illustrates how these fields are stored within the *MI*.

Entering a query into the Middleware Interpreter is accomplished by calls to two functions. The first, to

```
query = call Cloi.getQueryBuffer(int queryID, int
                 additionalSize),
```

which specifies the ID type of the query and the extra memory that must be allocated for the needs of the middleware or protocols, and that returns a pointer to the query. If the ID is already present in the *MI*, the returned buffer points to the address of the already registered query, and information may be overwritten. All fields with the exception of dtl and status, must be filled in the query before the query can be entered into the Middleware Interpreter. Invoking

```
Cloi.addQuery(MiddlewareQuery *query)
```

finishes the insertion of the query inside the *MI*.Access to a query is provided by read, add and remove functions, for instance

```
command MiddlewareQuery* Cloi.readInterest(int8_t queryID)
```

5.2.3 Composite Query Registration and Deregistration

The Middleware Solution must break complex queries into simple subqueries. Composite queries are entered with the composite field set to their query ID and by specifying the number of subqueries (this tells the Middleware Interpreter that it is part of a composite query). The Middleware Solution must enter subqueries immediately after with their `composite` field set to the ID of the composite query. The subqueries may be reused from already existing queries, and they can be part of other complex queries. A composite query is considered to have fired when all subqueries have fired (logical “AND” of the individual subqueries). The information is stored as shown in Table 5.2. In this example, the composite query with ID 6 denotes a fire (a high temperature, and a particular IR signature). It is made of two subqueries of ID 1 (temperature) and 2 (IR light).

The inverse operation of query deregistration is invoked with the following call:

```
call Cloi.removeQuery(int queryID).
```

For composite queries, the *MI* automatically searches for and removes all subqueries that are no longer used by any existing composite query.

Table 5.2: Composite Query Stored within the Middleware Interpreter

	Field	...	ID	Composite	subQueries	Status
Comp.	-1	...	6	**6**	**[1 2]**	IDLE
Sub.	TEMP.	...	1	**0**	**[6 5]**	IDLE
Sub.	IR	...	2	**0**	**6**	IDLE
Comp.	-1	...	5	**5**	**[1 3]**	IDLE
Sub.	PRES.	...	3	**0**	**5**	IDLE

5.2.4 Interest Registration and Deregistration

Protocols may register an interest in fields that are relevant to their behavior. An “interest” can be seen as a stripped down query that fires every time the field value is updated in the neighbor table.

The Middleware Interpreter can be tightly coupled to the neighbor table to follow what fields are of interest to the protocols in the stack. Interest in different types of queries is kept at the Middleware Interpreter, while other field (data or network related) interests are managed by the neighbor table. Upon updating a field in the neighbor table, *CLOI* checks whether it is of interest to any protocol. A positive answer results in signaling the change to subscribing protocols (if the value is different from the one previously stored). This procedure allows protocols to dynamically register for event notification, preventing unwanted events from interrupting the code when they are not needed.

5.2.5 Query Notification

A similar behavior is at work for queries. The burden of detecting that a query event has happened is now with the Middleware Interpreter, leaving the burden of data aggregation or query dissemination to the Middleware Solution. As a field is updated in the neighbor table (which includes information about the node itself, including sensed data values), the *MI* receives a notification from the neighbor table because it is a subscriber to all field changes. The *MI* matches the new field value to conditions expressed by registered queries and determines whether a query has fired. If so, the *MI* signals the event to all subscribing protocols, including necessarily the Middleware Solution and possibly other protocols down the stack. The notification identifies the query with its ID number and returns a pointer to the query in the Middleware Interpreter query table. This process is illustrated by Figure 5.2.

For composite queries, the *MI* looks through its entries and searches for the status of all its subqueries (which are identified in the `subQueries` field). If all have a *FIRED* status, the `composite` field of any subquery points to the aggregate query that will serve as the basis for the event notification. In case the composite query must be published to direct neighbors, the notification contains the updated values of all the fields related to the composite query, as well as the ID number of the query.

The Middleware Solution does not need to establish ad-hoc interfaces with other protocols to signal that a query has fired. In fact, it does not even need to use the *protocol event* signaling described in 4.4.4 as this is carried on by the *MI* automatically through the process described directly above. The notified protocols do not preoccupy themselves with checking whether an event is occurring, but they simply wait for event notification and then perform the appropriate actions when they are notified that a query has fired. For example, notification that a query has happened may prompt a routing protocol to refresh a route, a node activation protocol to wake-up neighbors, etc. When the query conditions are no longer met, a notification that the event has gone *IDLE* is sent and the protocols can similarly take action.

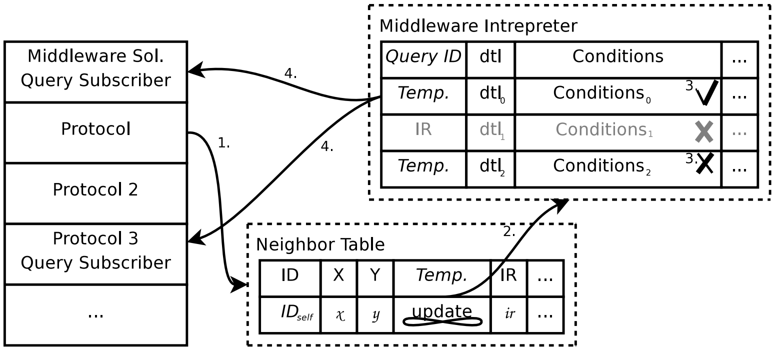


Figure 5.2: Event filtering and notification process: 1. A service or protocol updates the neighbor table. 2. If the field is of interest, and the new value is different from the old one, it is submitted to the *MI*. 3. The *MI* checks conditions realizing a query. 4. The *MI* notifies subscribing protocols and services that the query has fired.

Table 5.3: Query Notifications and Status Based on the Preexisting Status of a Query and Whether Its Conditions Are Met Upon an Update in the Neighbor Table

Conditions / *Status*	X	$\surd$
IDLE	—	Notify and set to FIRED
FIRED	Notify and set to IDLE	—

Table 5.3 summarizes the behavior of the *MI* based on the preexisting status of a query and whether its conditions are met.

5.3 Evaluation of Middleware Support

Although we cannot easily evaluate the convenience of this new middleware component in X-Lisa, it follows our goals of modularity, data freshness, and simplicity—the Middleware Interpreter is accessible by simply wiring to the existing **Cloi** and **CloiEvents** interfaces. However, we set out to evaluate what gains in QoS can be obtained through proactive query status notification to the protocols.

Before considering the gains, we note here the additional memory cost for providing proactive query status notification to the protocols. Using the Tmote Sky motes, the overhead is $7.4\ KB$ in the EEPROM and $100\ B$ in the RAM.

5.3.1 Middleware Interpreter Only

5.3.1.1 Health Monitoring Test Scenario

Let us consider a health monitoring application served by a sensor network where nodes are attached with heart rate and blood pressure sensors. As samples about the state of the patient are gathered, the application will accept varying quality of service depending on the aggregate stress level detected by the sensors. The fixed tree network topology is represented by Figure 5.3(a), where node 5 is monitoring the person of interest to the end user (all other nodes may act as routers). We assume that the node activation protocol keeps nodes 0 through 5 always active, and that the routing protocol has two routes from the source node 5 to the data sink 0: paths P_0, $\{5 - 4 - 2 - 0\}$ and P_1 $\{5$

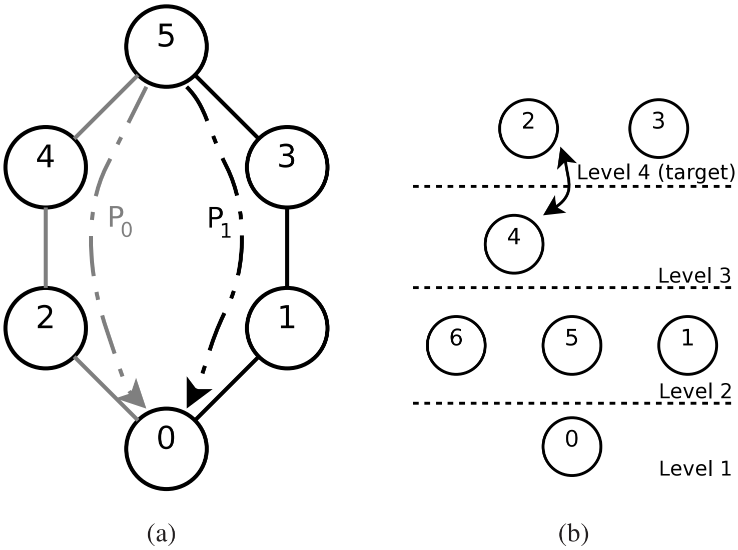


Figure 5.3: Topology of the (a) health monitor, and (b) room monitor test network.

- 3 - 1 - 0}. P_0 passes through nodes with higher residual energy, but suffers from higher packet error rates than P_1. Under high stress levels, packet delivery should be reliably sent to the data sink. The MAC protocol uses a Low-Power-Listening (*LPL*) scheme [18]: nodes sleep and periodically wake-up every $t_i\ s$ to listen for incoming transmissions. Packets must therefore be sent with very long preambles (at least $t_i\ s$) during which the destination will wake-up at least once. Most *LPL* MAC protocols, although very energy efficient, cause significant delays of ${}^{t_i}/_2\ s$ on average. In order to reduce delivery latency for urgent packets, we implement a *LPL* MAC protocol that may reduce its t_i value by a factor of four when certain conditions in the network and application are met.

The application requirements are summarized in Table 5.4. In our simulation, the patient's blood pressure increases from normal to high rapidly, and the heart rate readings cycle from low, to medium, to high, and again. Consequently, the source node will send packets with types 0, 2 and 4. Although this succession of sensor readings is not very likely in real deployments, it will help gauge the responsiveness of the network.

The simulation starts with the base station sending a query to spread its interest in detecting events related to the patient's health (source node 5). The routing and *LPL* MAC protocols subscribe to these queries (their type must be meaningful to all concerned layers). Every time a query state changes, notification of the routing protocol

Table 5.4: Application requirements for Middleware Interpreter only scenario

$BloodPressure\rightarrow$ $HeartRate\downarrow$	Normal	High
Low	30 s epoch Delivery Failure OK Long Delivery Delay OK Packet type 0	
Medium	30 s epoch Delivery Failure OK Long Delay OK Packet type 1	15 s epoch No Delivery Failure Long Delay OK Packet type 2
High	15 s epoch No Delivery Failure Long Delay OK Packet type 3	5 s epoch No Delivery Failure Short Delay Packet type 4

ensues, and it selects either path P_0 or P_1 (for the case *without* the *MI*, the routing protocol is left randomly choosing between P_0 and P_1 because it does not benefit from advanced query notifications). When a query event fires, notification is sent to the *LPL* MAC protocol, which increases or decreases its duty cycle by a factor of four depending on the current situation.

We simulated ten runs of this scenario for $1,800$ s in TOSSIM (the TinyOS simulator) and compared them to the protocol stack *without* X-Lisa or the Middleware Interpreter. While it makes no doubt that the application could be equally well served with ad-hoc middleware / other protocols interactions, that architecture would be exceedingly complex and inappropriate for modular designs. Thus, for fairness purposes, the *MI* solution was only compared to a layered scheme where protocols share a neighbor table among themselves, although it does not support event and query notification.

For each packet type (or patient state), we measure the per-hop delivery delays, the "output delay" (as the difference between a change in the patient's aggregate stress level and the time a report is sent), and total delivery delay (the time at which the data sink receives a packet whose type corresponds to the new patient state). Packet delivery ratios are also of interest and give a good indication of the possible QoS improvement brought by the *MI*.

5.3.1.2 Simulation Results

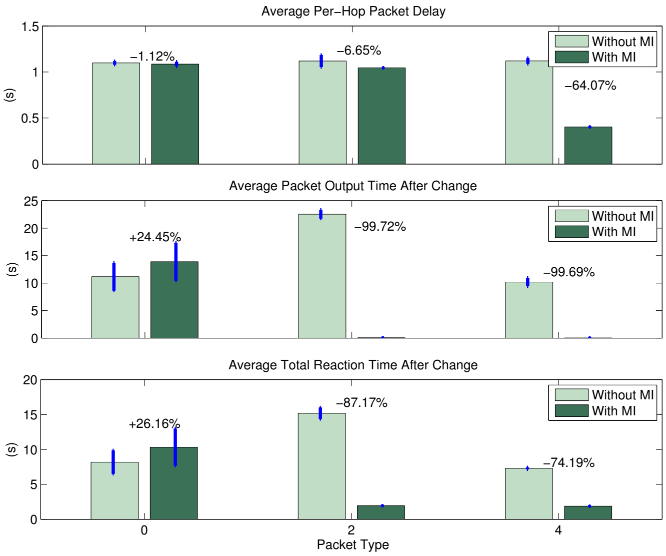
5.3.1.2.1 Delays Figure 5.4(a) shows the various delays experienced by different packet types. The per-hop delivery delay (top graph) illustrates the behavior of the MAC layer: for packet types 0 and 2, it is very similar for the case with and without the *MI*. However, type 4 packets enjoy a much reduced per-hop packet delay (at least 50% shorter). This is consistent with the requirements set by the application in Table 5.4.

With the *MI*, after a change in the state of the patient's health (middle graph), a packet is sent by the source after $15\ s$ for type 0 (no particular emergency), and only a few milliseconds in other cases. Without the *MI*, corresponding delays vary: this is because in a certain state, the application checks for the sensor readings periodically depending on the reporting epoch for that state. In state 0, the application checks the sensor reading every $30\ s$, causing the highest packet output delay when going from state 0 to 2. In state 2, the application reads sensor outputs every $15\ s$, which explains the delay when transitioning from 2 to 4.

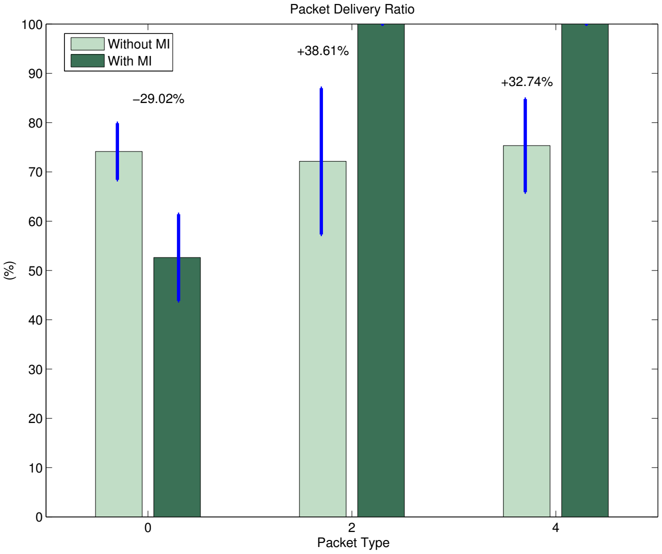
The total packet delivery delay after a change in the patient's state is shown in Figure 5.4(a), bottom graph. For the case when the patient is in a relaxed state (type 0), the *MI* actually increases packet delay because the application does not require immediate notification. In other cases however, The *MI* helps reduce the first state change notice packet to the data sink by 50 to 75% depending on the severity of the patient's health. This delay combines per-hop delivery gains as well as faster reaction with the *MI*.

These results show that both the application and MAC protocols can adapt to proactive query notification and substantially increase the QoS to the application.

5.3.1.2.2 Packet Delivery Ratios Figure 5.4(b) shows the benefits brought by the *MI* to the routing protocol and delivery reliability. Without the *MI*, packet delivery ratios stand at about 75% (the average of using a 100% reliable path P_1 and a $\approx 50\% = 0.8^3$ reliable path P_0). With the *MI*, delivery reliability of packets of type 0 is $\approx 50\%$ because the application can tolerate lower PDR on these packets, but 100% for packets



(a)



(b)

Figure 5.4: Comparison of the (a) delays (b) PDR for the patient monitoring scenario with and without the Middleware Interpreter.

denoting a higher patient stress level.

5.3.2 X-Lisa with Middleware Interpreter

In this section, we evaluate the Middleware Interpreter in the full context of X-Lisa. We use modified test scenarios, which we explain below.

5.3.2.1 Room Temperature Monitoring Test Scenarios

Let us consider a tree network in which nodes belong to levels. These levels can be thought of as different rooms or people in a building or a health monitoring application. Only nodes within the same or adjacent levels can communicate. The application is interested in monitoring the temperature in the highest level only—our "target" level (as shown in Figure 5.3(b)). If the temperature measured by a node exceeds a certain threshold, it must send a data packet (called urgent packet) every $2.5\ s$ until it changes levels, otherwise, a heartbeat data packet every $30\ s$ is sufficient.

We selected DAPR [6] as our node activation and routing protocol. DAPR assigns costs to nodes based on their importance to the application, and it selects nodes that can be spared as the optimal choices for node activation and routing. The MAC protocol is also a Low-Power-Listening (*LPL*) scheme. In order to reduce delivery latency for urgent packets, we implement an *LPL* MAC protocol that reduces its t_i value by a factor of four when a query fires.

The simulation starts with the base station sending a query to spread its interest in detecting events of high temperature in the target level. DAPR and the *LPL* MAC protocols subscribe to these queries. Every time a node changes location, notification of DAPR ensues, and it performs local route repairs. When a query event fires, notification is sent to the *LPL* MAC protocol, which increases its duty cycle by a factor of four.

In the first set of results, only one node is mobile. However, this node routes other nodes' packets, and it is capable of sensing temperature—and thus is a source of data packets as well as a relay. As a consequence, its movement among levels is highly disruptive to DAPR, which was not originally created for a mobile network topology. In the second set of results, half the nodes are mobile, and we added one more level.

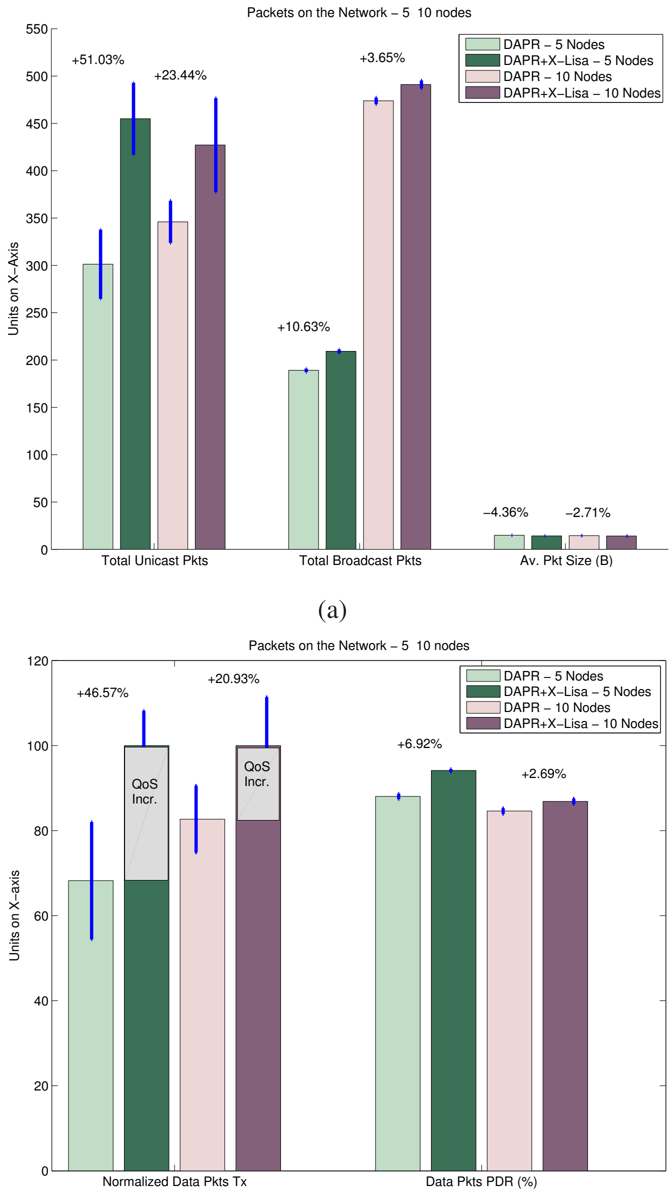
We simulated these scenarios with five and ten nodes for $1,800\ s$ in TOSSIM (the TinyOS simulator) and compared them to the protocol stack *without* X-Lisa with middleware support. For fairness purposes, the protocol stack without X-Lisa also shares a neighbor table among layers, although it does not support event notification. The results are averaged over ten runs each.

We measure delays between a change in level and the start of the temperature reporting and the delivery delay of regular and urgent packets. Packet delivery ratios are also of interest and give a good indication of the possible QoS improvement brought by X-Lisa with middleware support.

5.3.2.2 Low Node Mobility

5.3.2.2.1 Packets Exchanged on the Network Figure 5.5(a) characterizes the packets traveling on the network. Broadcast packets can be seen as the network overhead—DAPR and X-Lisa send them to manage the network. The increase in the total number of broadcast packets when X-Lisa is used is around 10%. With this increase, we observe a large rise in unicast packets (whether heartbeat packets or "urgent" data packets) for X-Lisa of 25% to 50%. These two measurements already show a significant increase in QoS, at the price of a limited increase in overhead represented by the broadcast packets. With X-Lisa, the average packet size is smaller thanks to the update strategy that limits the content of the information vector to only the fields that have changed, while without X-Lisa, all fields must be sent at the beginning of every DAPR round. Note that the packet size would only be of limited importance for *LPL* MAC protocols since larger packets do not lead to higher energy consumption. However, other MAC protocols may consume more energy for larger packets, and the smaller average packet size of X-Lisa would help limit the increased overhead.

Figure 5.5(b) shows that the increase in total unicast packet transmissions stems from a larger number of packets being originated in the case of X-Lisa—and not an increase in retransmissions for instance. The large increase in unicast packets reflects an improvement in QoS. Here, the mobile node moves in and out of the target level. As soon as it starts meeting the query criteria (matching level and sensor reading conditions), it can send report packets. However, without X-Lisa, a topology change goes unnoticed until the natural route rediscovery period. Many more packets are sent and received because X-Lisa is capable of notifying all layers in the stack of topology changes.



(a)

(b)

Figure 5.5: Comparison of the number and nature of packets on the network for the low mobility scenario with and without X-Lisa.

5.3.2.2.2 Delays Figure 5.6(a) presents various delays of interest that suggest improved QoS provided by the network. The average neighbor notification delay is the time between a change in the topology (a node moving up or down a level), and the time all its new and former neighbors learn of it. This measure typically reflects how accurate the local vision of the network is. X-Lisa proactively notifies neighbors that a change has happened so as to maintain up-to-date routes. In the absence of X-Lisa, routes are updated every DAPR round ($60\ s$) during which route discovery phases are implemented. A long neighbor notification delay causes loops and packets to be misrouted. The improvement brought by X-Lisa with Middleware Support is drastic (well above 80%).

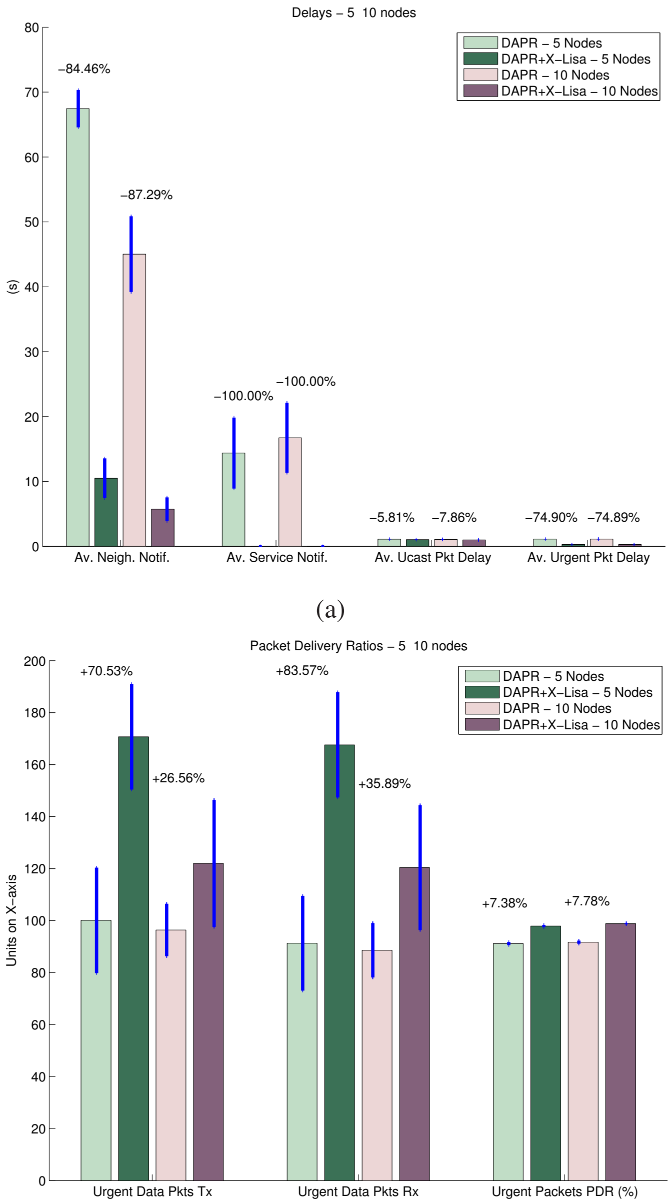
The average service notification delay is the time between a node's arrival in the highest level and the start of the temperature service. With X-Lisa, there is no delay as the detection is immediately followed by the notification. Conversely, without X-Lisa, the application must periodically inquire whether the node is in the target level (every $30\ s$ since this is how often a heartbeat data packet is sent under normal circumstances) and if so, it checks for the node temperature in the neighbor table (reasonably, every $2.5\ s$).

The average regular and urgent data packets delivery delays measure the difference between the time a packet is scheduled to be sent and when it is received at its destination. For regular packets, the difference with and without X-Lisa is small: the MAC layer's *LPL* scheme does not reduce its inter-listening time t_i and we only observe that malformed routes are responsible for the slightly superior delivery delay without X-Lisa. However, X-Lisa allows the MAC layer to benefit from the knowledge that a query has happened, and that its epoch time requires the t_i value to be reduced. This results in an improvement of $^3/_4$[1], as the division of t_i by four would suggest.

5.3.2.2.3 Packet Delivery Ratios Figure 5.6(b) shows the packet delivery ratios for regular and urgent data packets. In general, the packet delivery ratio for X-Lisa is slightly greater than that without it (by 2.5% to 7%). However, the packet delivery ratio for urgent data is significantly higher, by 7%. The increase in urgent data packets sent over the network with X-Lisa is above 70%. In spite of the increased traffic, the more accurate network vision provided by X-Lisa helped guide more packets to their destination.

---

[1] A command in the first packet prompts all nodes to reduce their duty cycle.



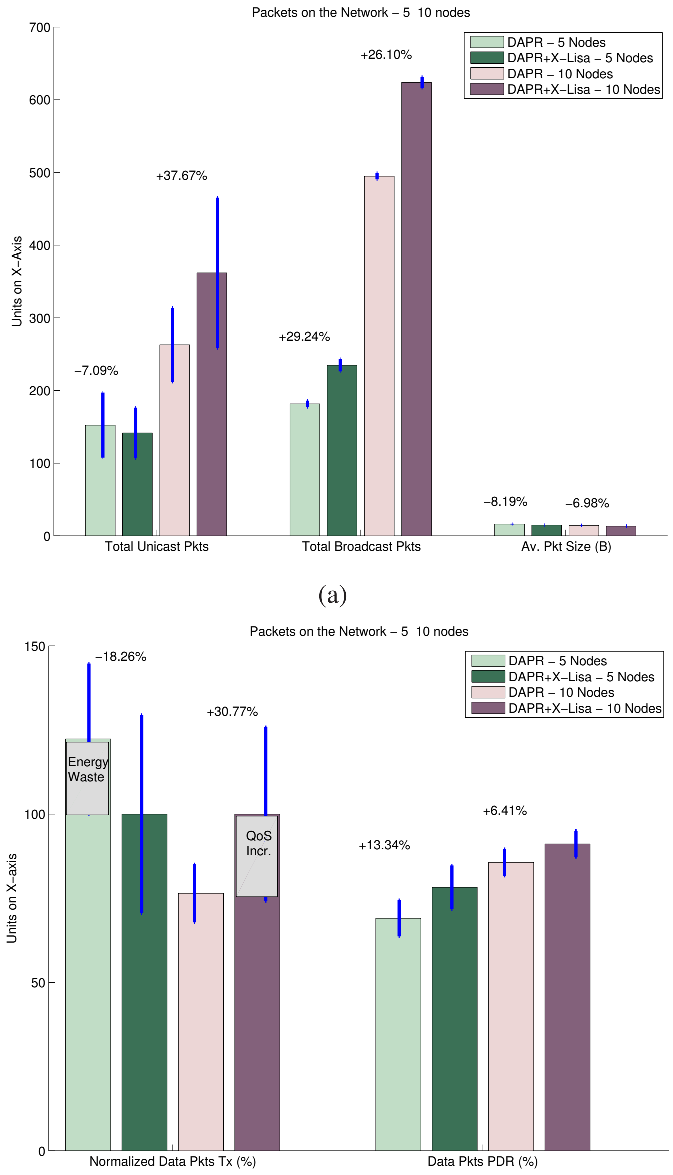
(a)

(b)

Figure 5.6: Comparison of the (a) delays and (b) packet delivery ratios for the room monitoring scenario with low node mobility with and without X-Lisa.

5.3.2.3 High Node Mobility

In this section, half the nodes of the tested scenario are mobile. This puts a strain on the routing and node activation protocols that must keep routes up-to-date.



(a)

(b)

Figure 5.7: Comparison of the number and nature of the packets on the network for the room monitoring scenario with high node mobility with and without X-Lisa.

5.3.2.3.1 Packets Exchanged on the Network Figure 5.7(a) significantly differs from the first scenario in the number of unicast packets sent for five nodes. The number of unicast packets sent is not necessarily higher when X-Lisa is used because the application node will stop sending heartbeat or urgent data packets as soon as a change in level is detected. This change may go unnoticed if X-Lisa is not used, causing extra unicast packets to be sent. In this case, the energy to send unnecessary packets is wasted and does not translate into increased QoS because the information sent is stale. Figure 5.7(b) shows the same results.

5.3.2.3.2 Delays Figure 5.8(a) shows that the neighbor notification delay for both cases has increased compared to the first scenario. This is mainly due to the fact that more neighbors of a moving node will be out of range when the notification is sent, causing them to ignore many changes until the next DAPR round or until X-Lisa automatically removes stale entries.

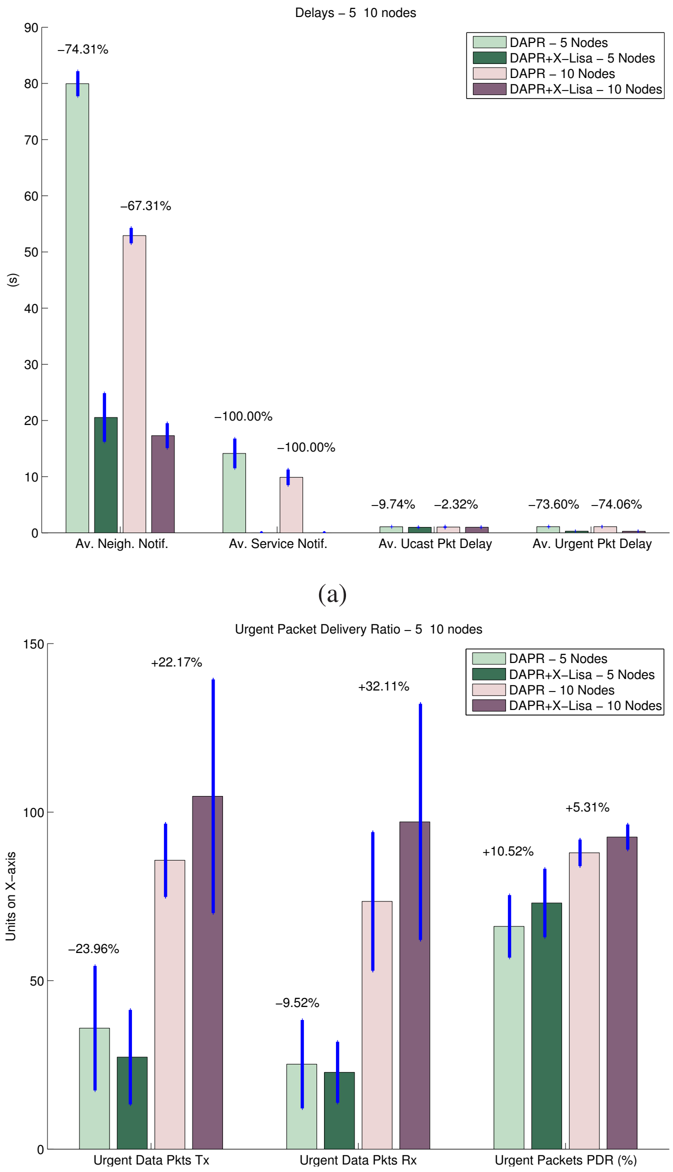
The average service notification delay is similar to what was observed previously, as well as the unicast packet and urgent packet delivery delays. In spite of increased mobility in the network, X-Lisa obtained the same reduction in urgent packet delivery delays (75%) over DAPR only.

5.3.2.3.3 Packet Delivery Ratios The delivery ratio for regular packets shows a significant increase as the network goes from five to ten nodes, and if X-Lisa is used, as seen in Figure 5.8(b). As the number of nodes increases, more routes are available to route packets, and it is thus reasonable to see an increase in PDR. However, X-Lisa is capable of delivering up to 13% more packets through the maintenance of accurate routes. The effect is felt more when fewer nodes are available to form these routes. The results for urgent packets PDR is very similar to that of regular packets.

The number of sent and received urgent packets may be reduced because a node can more easily identify that it has moved out of the target level. This happens here for the simulation of five nodes.

5.4 Summary

WSN architectures should exhibit some important features in order to boost their popularity, most importantly support for cross-layer protocols and modularity. In this chapter, we propose extending these principles to the middleware solution. A Middleware Intrepreter component extends the benefit of proactive event notification to all protocols, vertically (through layers of a same node) and horizontally (between nodes of



(a)

(b)

Figure 5.8: Comparison of the (a) delays and (b) packet delivery ratios for the room monitoring scenario with node high mobility with and without X-Lisa.

the same neighborhood). The Middleware Interpreter provides a convenient information repository for application queries and their management. We added these ideas to X-Lisa, a cross-layer information-sharing architecture that retains a layered structure while supporting cross-layer improvements.

The originality of this work lies in the event filtering made possible by X-Lisa: in many cases, protocols need not be aware that a new sensor reading is available, only that an application query has occurred. For instance, the routing protocol may update its routes, and the node activation protocol may choose to wake up more neighbors.

We implemented these new ideas in TinyOS so as not to ignore limitations imposed by some lightweight operating systems. Simulation results showed up to 13% increase in packet delivery ratios, even as the number of data packets delivered to the data sink could better fit the needs of the application and the reality of the network. X-Lisa was also successful in reducing urgent packet delivery delay. These improvements (of up to 75% in packet delivery delays for the tested scenarios) were obtained with a limited increase in overhead (of 3% to 30% in broadcast packets).

Chapter 6

Middleware for Supporting Protocol Adaptation

Chapter 3 demonstrated the existence of tunable parameters (*knobs*) within cross-layer protocols. These may be tweaked to adapt the response of the network to specific needs of the application. This guarantees, for instance, that nodes that will be needed during critical stages of an alert do not waste energy during normal states of the network. This chapter introduces how middleware is able to provide application-level information to the network protocols, enabling them to provide adequate service to the application for longer periods of time.

6.1 Introduction

In order to increase the effectiveness of cross-layer architectures for different applications, oftentimes these architectures have several parameters whose value can be tuned to better serve each specific application. However, it would not be reasonable to expect the application itself to control these different parameters—this is better handled through an intelligent middleware. The middleware's task is to coordinate application needs to quality of service rendered by the network, using commands understandable by the protocol.

Here, we show how a sensor network middleware can meet this requirement by appropriately managing a cross-layer sensor network protocol. We base our work on the cross-layer protocol DAPR (Distributed Activation with Predetermined Routes) [6] and the sensor network middleware MiLAN (Middleware Linking Applications and Networks) [53].

6.2 MiLAN: a Sensor Network Middleware

This work builds off MiLAN, which is introduced here in more details. Several sensor network applications require that only certain sensors be activated at a given time, and that this set of required sensors change over time. For example, a home / office security application may require that only entries to a room be monitored during normal operation mode, whereas different sensors are needed when an intrusion is detected. Similarly, a health monitoring system may require only a few vital signs to be monitored while the individual is healthy, but when signs of stress are detected, more vital sign monitoring is needed.

Controlling sensors for such dynamic operation, where the set of active sensors depends not only on the sensors themselves (remaining energy, location, etc.) but also on the state of the application and the system being monitored, is difficult to do directly from the application. MiLAN was designed to ease this burden on the application designer by incorporating all the necessary mechanisms for controlling the network within the middleware [53]. MiLAN provides the application an application programming interface (API) through which it can specify its quality of service (QoS) goals, and MiLAN takes care of appropriately setting network parameters to meet these QoS goals over time. Large gains in lifetime can be obtained by allowing MiLAN to adapt a set of tunable network parameters over time to just meet the application's QoS requirements.

We assume that the quality of different sensed data can be quantitatively evaluated. In order to configure the sensors and the network parameters to meet application needs, MiLAN must know (1) the variables of interest to the application, (2) the required QoS for each variable, and (3) the quantitative QoS of data output by sensors. All of these quantities may vary over time. This information is conveyed to MiLAN through "State-based Variable Requirements" and "Sensor QoS" graphs, as shown in Figure 6.1. Using this information, MiLAN configures the sensors and begins the normal network operation, where sensors send data to the application.

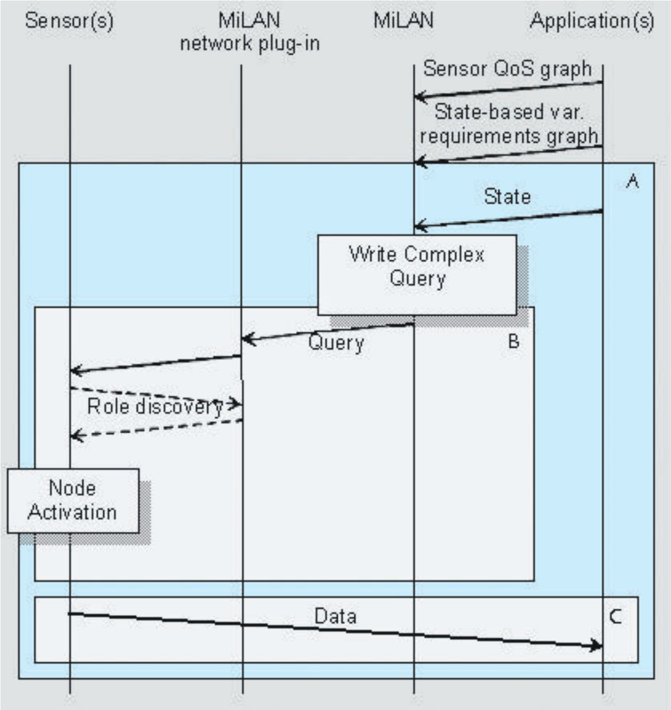


Figure 6.1: Overview of MiLAN. A and B start when the application starts or changes its state depending on data received from the sensors. C is the normal mode of operation when conveying the sensors data to the application.

The following section discusses how to use MiLAN to control the operation of DAPR, thereby improving overall lifetime of the network and easing the application of the burden of sensor and network management.

6.3 Managing DAPR Through MiLAN

Previous work on MiLAN focused solely on one-hop networks using standard, layered protocol architectures (*e.g.*, Bluetooth, IEEE 802.11). Multi-hop heterogeneous networks and cross-layer architectures offer new challenges to MiLAN. We believe that MiLAN can exploit the tunable parameters of a cross-layer protocol like DAPR (see Chapter 3 for a full description of DAPR) to save energy while meeting application QoS for such multi-hop networks.

6.3.1 Overview of MiLAN/ DAPR Combination

Using MiLAN to orchestrate a network running DAPR will provide many advantages in terms of extending network lifetime while meeting dynamic QoS constraints. Parameters of DAPR such as application cost definition and query interval can be adapted to provide maximum benefit to the application. To effectively manage DAPR, MiLAN must obtain the application's current QoS requirements as well as the system state. This global view of the application is used to create queries through which MiLAN communicates with DAPR, as depicted in Figures 6.1 and 6.2.

6.3.1.1 Complex Queries

MiLAN has a clear view of the application's required QoS at every point in time. Its main goal is thus to issue queries understandable by DAPR to meet the QoS goal while minimizing energy dissipation. However, new and more sophisticated applications will require *complex* queries. Such information as the monitored variables, the required precision (also called confidence) for each variable of interest, the area of interest in the network, and the reporting mode (*continuous* or *discrete*) need to be included in these queries.

6.3.1.2 Variables and Precisions

MiLAN was designed to manage heterogeneous sensor networks. Variables correspond to physical attributes (*e.g.*, temperature, intruder presence) that can be described by XML tags. Each node should know which variable it can monitor and with what precision prior to deployment.

6.3.1.3 Relative Weight of Required Precision

MiLAN also has global information on potential or future application requirements. For instance, it may be readily known to MiLAN that a precision of $conf(v_j)$ on variable v_j is never needed. We propose using a vector P^j of weights or probabilities inferred from the various graphs inherited from the application that indicates the relative importance of each precision $conf(v_j)$ for variable v_j. For simplicity, we quantize $conf(v_j)$ by 10%, and thus P^j indicates how often the variable v_j is required in different precisions from 0.0 to 1.0 in steps of 0.1. For instance, it may be known that variable v_j requires a precision of 0.6 50% of the time and 0.7 50% of the time, with the other precisions not required by the application. Thus P^j would be [0 0 0 0 0 .5 0 .5 0 0 0].

6.3.1.4 Relative Weight of Variables

In addition, MiLAN's knowledge of the potential QoS requirements can lead to the conclusion that a variable will never be needed, or that a variable is extremely important in all application states. Thus, MiLAN also issues a vector W that weights all the variables in the system. W and P help assess the relative importance of all sensors to aim at conserving the most critical ones.

6.3.1.5 Entities within the Monitored Region

The network can also be divided into entities or groups of sensors. An entity can be described by an XML tag such as *windows*, *soldier A*, or *second floor*. If the network extends over large distances (as is possible in the case of military applications, for instance), chances are that several sensors monitoring the same variable may be able to respond to a query to meet the required QoS, but that only a subset of these will be within the entity that is truly of interest. Furthermore, two sensors in range of each other (*e.g.*, two sensors on different soldiers) may be monitoring two different entities and thus be incapable of cooperating on their sensed data.

We contend that each sensor needs to know to which group of sensors it belongs, using high-level semantic descriptions. For practical reasons, we propose that all sensors are attributed a tag, that is shared with other sensors monitoring the same entity. MiLAN can use this tag to specify the entity of interest in the queries.

6.3.1.6 Setting the Query Interval

The query interval is a critical factor in the protocol overhead, and thus in the network lifetime. While an application might need to frequently change the active sensors because of changing QoS requirements, a small query interval degrades the network lifetime. Moreover, for an application that requires full network coverage, while a frequent query update can preserve 100% coverage for longer times, it leads to a quicker mid to low range coverage degradation later in time.

Intuitively, we can see that application states that correspond to slowly changing situations probably do not require frequent network changes and thus long query intervals are desirable. Once a more critical state is reached, such as a high stress state in a medical application, a smaller query interval is likely to be used to meet possibly rapidly changing application QoS. On the other hand, if the maximum attainable QoS is already significantly degraded, the query interval should be made longer to increase the network lifetime by lowering the overhead.

6.3.1.7 Form of the Complex Queries

The DAPR queries must include the reporting mode (*continuous* or *discrete*), frequency of reports from the sensors, required precisions of variables of interest, the precision weight vector P, the variable weight vector W, and the round length V_s. Also included in the DAPR queries is the cumulative application cost (the sum of the costs of the reverse path back to the data sink) to create routes as the queries are disseminated throughout the network.

6.3.2 Routing: New Variable-based Cost

For complex, non-coverage-type applications that require the network to monitor one or more variables, any group of sensors that provides data about each variable with greater than the required precision (confidence) provides 100% utility or QoS. In some cases, only one sensor's data may be enough to provide 100% QoS. For these new variable-based applications, the application cost in DAPR should evaluate the relative importance of a node with regard to each variable. Thus, for this new variable-based QoS metric, we define application cost for sensor k as:

$$C_k = \sum_{j=1}^{V} W_j \cdot Cv_{j,k} \tag{6.1}$$

where V is the number of variables the network is capable of monitoring, W_j is the weight of variable v_j and the per-variable cost is:

$$Cv_{j,k} = \frac{[\sum_{i=1}^{10} P_i^j \text{ such that conf}(v_{j,k}) \geq P_i^j] \cdot \frac{1}{E(k)}}{\sum_{m=1}^{N} [\sum_{i=1}^{10} P_i^j \text{ such that conf}(v_{j,m}) \geq P_i^j] \cdot \frac{1}{E(m)}} \tag{6.2}$$

where P^j is the probability vector (P_i^j is the i^{th} element of P^j), *conf*$(v_{j,k})$ is the confidence with which sensor k can monitor variable v_j, and $E(k)$ is the remaining energy of sensor k. C_k evaluates the attention given to each variable and divides the precision of sensor k by the sum of all precisions across all the sensors in the monitored entity. The normalized inverse of the remaining energy is included to differentiate between nodes whose remaining energy is very low but are critical to the application and other sensors whose energy is still high. The sum in the denominator is not an algebraic sum of all precisions *per se*, but rather an evaluation of all the energy devoted to monitoring a variable with various precisions or confidences.

Table 6.1 gives a concrete example of application cost calculations.

6.3.3 Node Activation: a Distributed Process

New challenges in a multi-hop network, as well as the specificities of DAPR, require that the node activation process be distributed. DAPR's node deactivation process ensures that a node will deactivate only if the application requirements are met by other active sensors; otherwise, if its own precision is within the tolerance of the required precision and no other lower cost sensor can provide this data, the node will stay active. Nodes consider deactivating in the order inverse to their cost. Figure 6.2 illustrates this process.

Immediately after this procedure, the selected nodes send a data packet to the sink using their pre-computed smallest cumulative cost path. Only the activated sensors and those contributing to the return path (those that retransmit the packet) will remain active in the round.

6.4 Results

This section presents results that will show the merits of using MiLAN to manage DAPR over a simpler architecture that aims to serve a fixed (often the strictest) QoS.

Table 6.1: Detailed calculation of the application costs for 4 sensors and 2 variables. The final costs reflect that the same importance is granted to a precision of 0.5 and 0.9—the application has no use for a precision higher than 0.5 and lower than 1.0. The tables at the bottom give the values of P and W.

Sen. Var.	S$_1$	S$_2$	S$_3$	S$_4$
V_1	0.5	0.9	1.0	0.0
V_2	0.0	0.0	0.0	0.9
C_{V_1}	$\frac{0.3+0.1+0.1}{2.0}$	$\frac{0.3+0.1+0.1}{2.0}$	$\frac{0.5+0.5+0.3+0.1}{2.0}$	$\frac{0.0}{2.0}$
C_{V_2}	$\frac{0.0}{1.0}$	$\frac{0.0}{1.0}$	$\frac{0.0}{1.0}$	$\frac{0.5+0.5\times0.1}{1.0}$
Cost	$0.5\times\frac{0.5}{2.0}+0.5\times0$ $=0.125$	$0.5\times\frac{0.5}{2.0}+0.5\times0$ $=0.125$	$0.5\times\frac{1.0}{2.0}+0.5\times0$ $=0.25$	$0.5\times0+0.5\times\frac{1.0}{1.0}$ $=0.50$

P			P		
C	V_1	V_2	C	V_1	V_2
1.0	0.5	0.0	0.5	0.3	0.0
0.9	0.0	0.0	0.4	0.1	0.1
0.8	0.0	0.5	0.3	0.1	0.1
0.7	0.0	0.0	0.2	0.0	0.1
0.6	0.0	0.0	0.1	0.0	0.1

	W
V_1	0.5
V_2	0.5

The simulations are carried out in Matlab and use one-hop networks to determine the advantage of intelligent selection of the active sensors based on varying QoS requirements[1]. A fixed amount of energy per second is used by the active nodes (0.1% of the total initial energy).

---

[1] Previous work on DAPR [6] has already shown the positive effects of application cost on energy savings for routing.

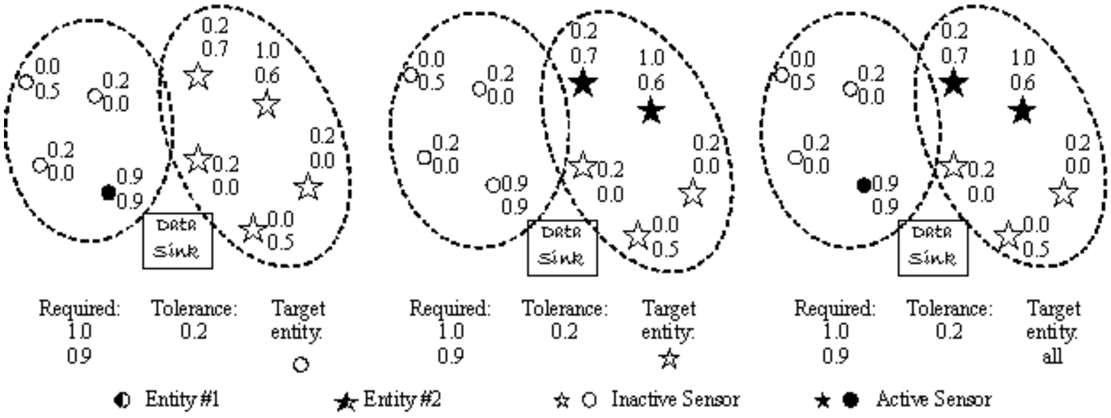


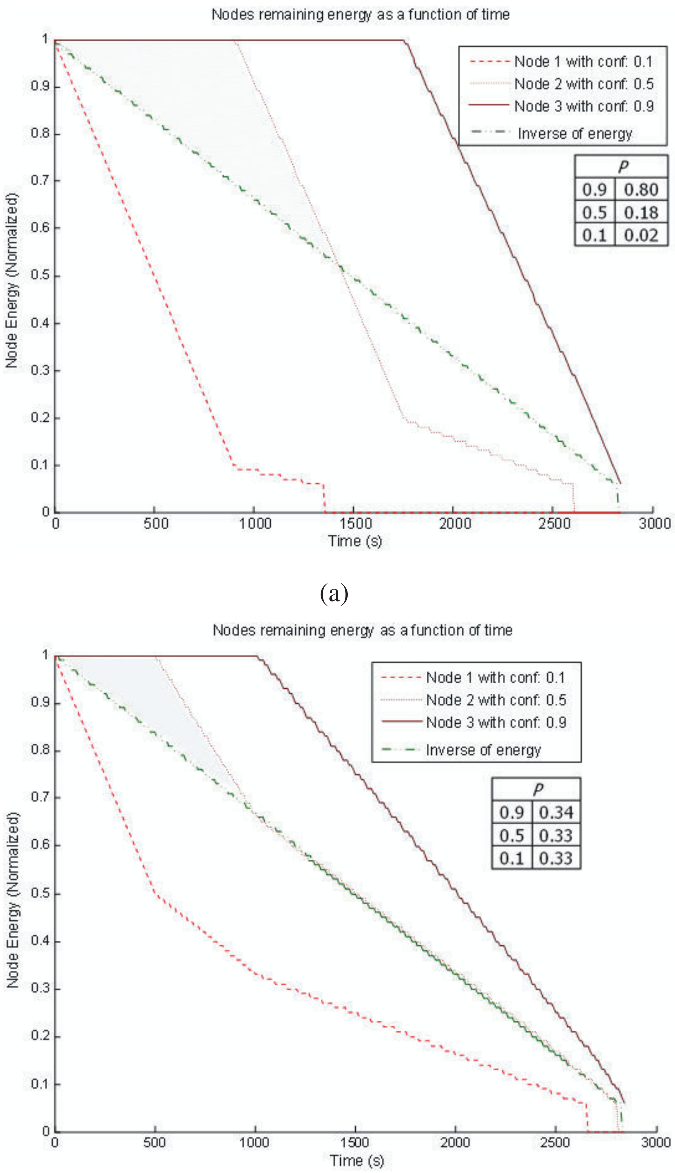
Figure 6.2: Nine sensors are monitoring two different entities (*e.g.*, two different soldiers in range of one another). The required precisions for the two variables of interest are 1.0 and 0.9, with a tolerance of 0.2. Three cases are possible: only the first entity (with sensors represented by circles), only the second entity (with sensors represented by stars) or both entities are of interest.

6.4.1 One Variable with Low QoS Requirements

We first compare our application cost to the inverse of the energy cost in order to validate our choice of application cost. For this simulation, three sensors, each with the same initial energy, are able to monitor one variable with precisions 0.1, 0.5, and 0.9 for sensors 1, 2, and 3, respectively. The only QoS requirement is a non-zero precision, *i.e.*, some idea of the measured variable is all that is needed.

Figure 6.3 shows the remaining energy of each node using application cost and the inverse of remaining energy cost for a given P vector. The sensors' energy decreases by the same amount when the cost is the inverse of the energy since all nodes are eligible to be picked for activation, and thus they are selected in a round-robin manner. Conversely, application cost allows the energy of the most important sensor (with QoS 0.9) to be spared for longer. When sensor 1 dies, sensors 2 and 3 have half and full remaining energies; when sensor 2 dies, sensor 3 has a third of its energy remaining. Application cost would allow for better QoS as illustrated by the shaded area, were the application to request it.

The effect of P is not trivial: when a higher weight is assigned to high precisions, node 3 is spared for longer than when all weights are the same, at the disadvantage of



(a)

(b)

Figure 6.3: Node remaining energy over time. All three nodes are able to monitor a single variable with precisions 0.1, 0.5, and 0.9. The QoS requirement is 0.1. a) and b) have different P vectors.

the remaining sensors. It may be good to conserve the energy of important sensors, but this could be unnecessary in some applications where high precision is never needed. We believe statistical information on the application can help find a suitable P.

These results show the merits of our application cost. However, both application

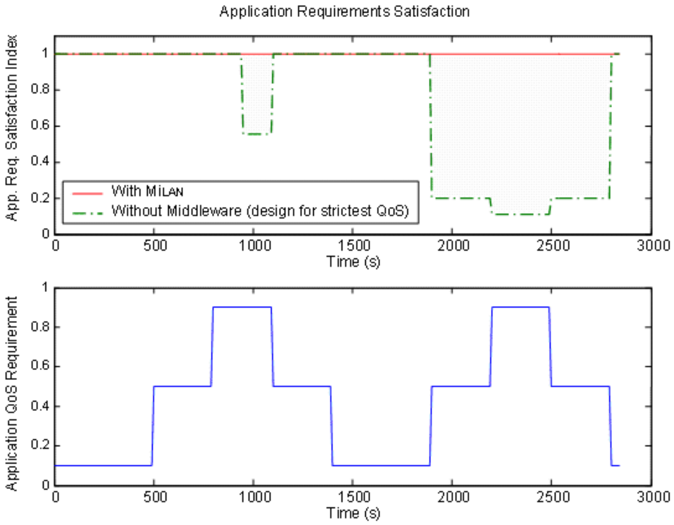


Figure 6.4: A measure of the QoS differences between a system with MiLAN managing DAPR and a system designed for the strictest QoS. The graph at the bottom shows the application QoS requirement.

cost and the inverse of energy cost bring a significant improvement compared to a system architecture that does not use a middleware to manage the network. To understand the full improvements of our design, we need to evaluate the lifetime and QoS for a network that does not have the benefit of a middleware, and thus simply selects sensors providing the highest QoS available in the network.

6.4.2 QoS Increase with a Middleware

The conditions of these simulations are similar to the previous experiments. We compare our system design (MiLAN managing DAPR) to a system that has no smart middleware. The latter architecture is only able to activate nodes that will provide the highest QoS available in the network and does not adapt the network QoS to the changing application needs. To quantify the gains in QoS, we calculate an index that averages the percentage of requirement satisfaction. For instance, if the application requires a precision of 0.9 but the network can only provide 0.5, the index is equal to $^{0.5}/_{0.9}$. Obviously, this QoS index has no concrete significance since the precisions are not linear; however, we believe it provides a good basis for comparison.

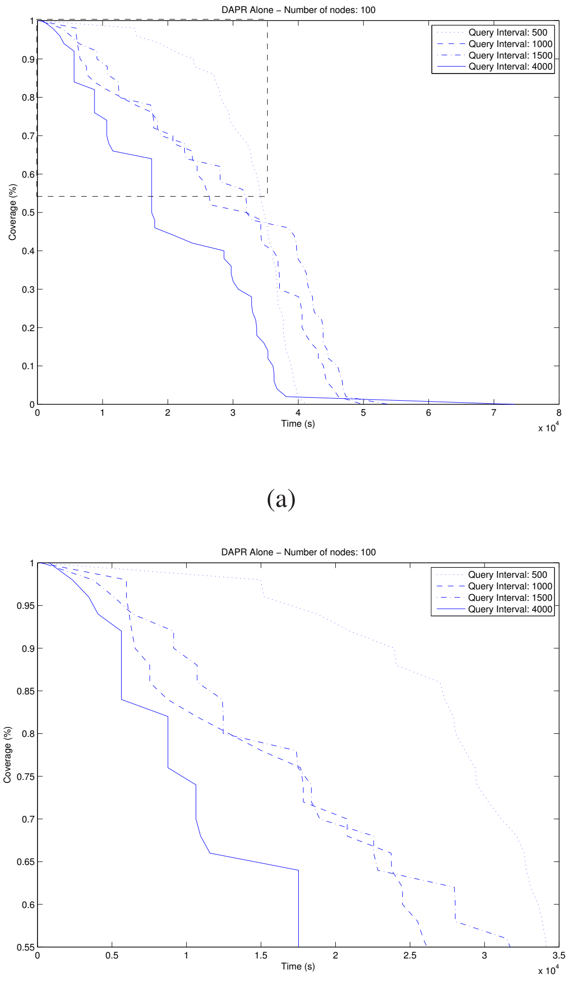
Figure 6.4 shows this QoS index during the simulation time (top graph). MiLAN leads to significant improvements by adapting the network QoS to that of the application (bottom graph). On the other hand, the simpler design that selects the sensor with the highest precision at all times uses the energy of high precision or scarce nodes first, even when this is not necessary. During the simulation time, our architecture is always able to provide nominal QoS, while the system with no middleware support offers poor precision during many of the critical stages where a maximum QoS is needed. The shaded areas represent the QoS gains ascribable to MiLAN.

6.4.3 Change of Query Interval

The notion of complex queries introduced in Section 6.3.1.7 also involves a changing query interval that responds to the needs of the application and the conditions on the network. For fixed queries, the interval between two consecutive route discovery phases is set prior to deployment, and since it should fit the whole network, it has to be a conservative value. However, one query interval may not fit all scenarios indiscriminately. In this section we prove this concept through the simulation of a sensor network whose primary objective is to monitor a region.

The simulations were carried out in NS-2.28 without any input from MiLAN, for various but fixed values of the query interval and for continuous monitoring. The lifetime at $x_L\%$ ($x_F\%$) is defined as the last (first) time when the network could (no longer) provide coverage of at least x percent. In Figure 6.5, the lifetime is defined as the first time when the network failed to provide the required coverage as would be asked for the monitoring of sensitive property like a financial institution. Figure 6.5(a) shows that for the mid-to-low range coverage (less than $55_F\%$), a longer query interval (1,500s) increases the network lifetime compared to other values. On the other hand, Figure 6.5(b) presents the same results for high range coverage. A query interval of 500s yields longer lifetimes at $100_L\%$ to $55_L\%$ coverage by as much as 400% over the longest query interval. In Figure 6.6, the application is such that the end user can tolerate short periods of time ($<$ 2 query length intervals) when the network is incapable of monitoring at least $x_L\%$ of the region. We draw the same conclusions as previously, although for higher-range coverage ($> 95\%$).

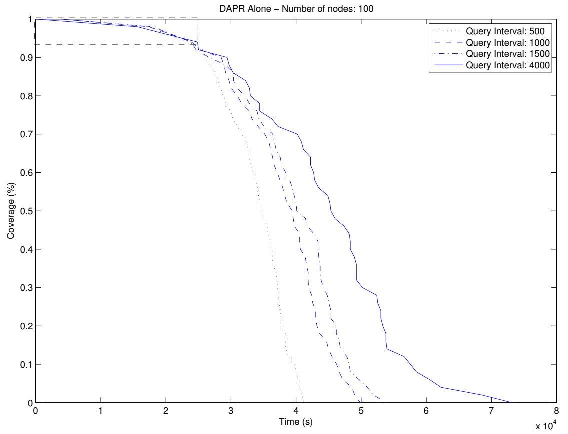
When queries are sent frequently, the cross-layer design wields its full optimization and energy savings characteristics. This results in a longer lifetime. Figures 6.5 and 6.6



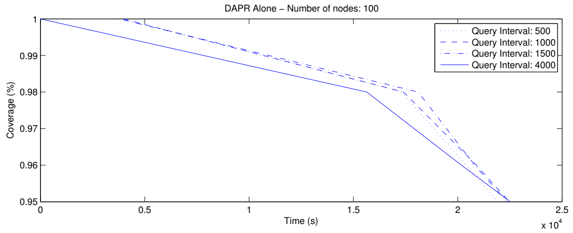
(a)

(b)

Figure 6.5: (a) Lifetime of the network as defined by the time when the network was first incapable of providing a coverage of $x_F\%$ for different but fixed values of the query length interval. (b) zooms on the results for high-range coverage.



(a)



(b)

Figure 6.6: (a) Lifetime of the network as defined by the time when the network was last capable of providing a coverage of $x_L\%$ for different but fixed values of the query Length interval. (b) zooms on the results for high-range coverage.

illustrate this characteristic toss-up and show that a middleware can adapt the query interval to the application needs as well as to the current coverage level in the network in order to maximize the lifetime as defined by the application. From Figures 6.5(b) and 6.6(b), we can infer that a good query interval for high QoS would be between 500s and 1,500s, while a longer interval may be desirable when the maximum QoS is not achievable.

Figure 6.7 presents a comparison of simulations with an adaptive and a fixed query interval from the perspective of QoS delivery. In these simulations, two sensors are able to monitor a variable with precisions 0.5 and 0.9. The application requires precisions of 0.5, 0.9, or has no interest in the variable, as shown in Figure 6.7 (c). In the adaptive query interval scheme, the QoS provided by the network is submitted to “gravity”: when the application QoS requirement increases, MiLAN waits for it to be constant for

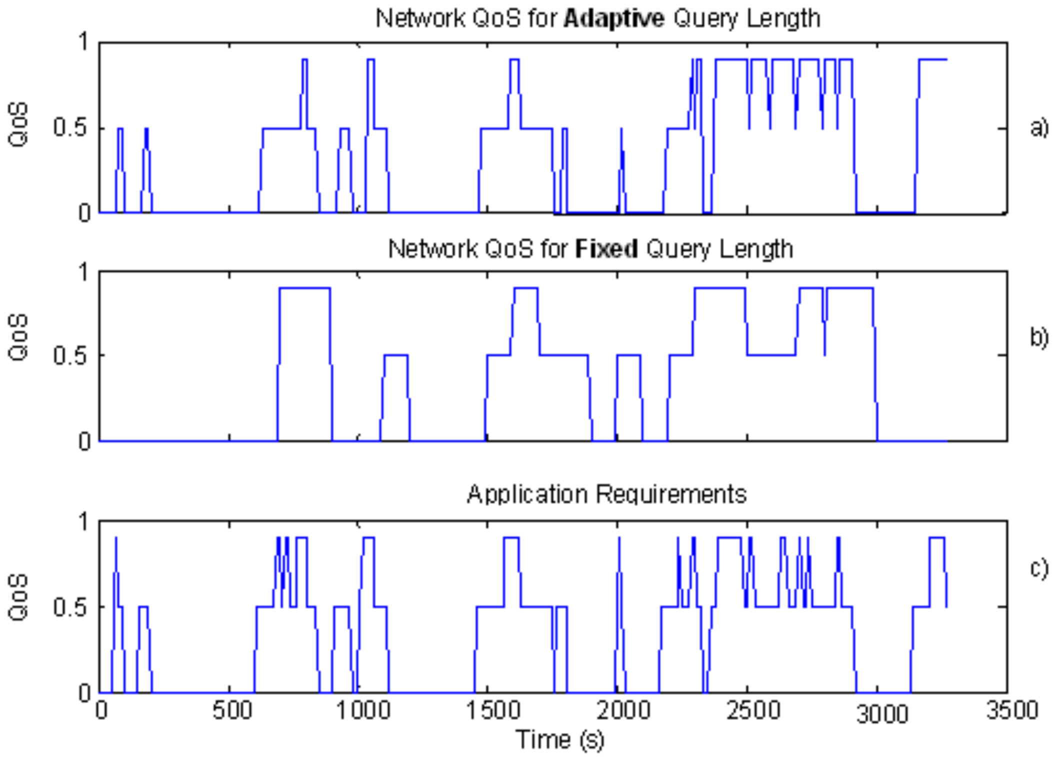


Figure 6.7: Network QoS response of the adaptive (a) and fixed (b) query length schemes to the application needs (c).

30s (a "weight") before it issues a new query. Conversely, when the QoS requirement decreases, MiLAN sends a query immediately. In the fixed scheme, the query interval is set to 100s. Every 100s, MiLAN checks for the current application requirement and floods a query with this information. The network will try to meet the QoS for the remaining query interval. In both cases, flooding a query costs the same amount of energy, 0.5% of the initial energy in our simulations.

Figure 6.7 (a) and (b) present the QoS provided by the network in both schemes. Figure 6.7 (b) shows that much of the energy is wasted when the query is issued at the time of a QoS application requirement peak. The network can also miss QoS requirements for up to 90s. In contrast, an adaptive query length design is able to respond to an increase in the application need within its 30s weight. Consequently, the QoS index for the adaptive query length scheme is 40% higher than that of the fixed query interval scheme.

Some applications may need an immediate response from the network. We simulated an adaptive query length scheme with no gravitational pull and found that the application requirements are always met, but for a shorter time—the network lifetime is reduced by about 10%.

6.5 SRI: A General Cost Function

Because it is difficult to map the value of vector P to actual application requirements, SRI (Sensor Replacement Index) was introduced in [76] to differentiate between sensors. SRI helps quantify the importance of a node to the application so that critical sensors may be alive and available when they are most needed.

SRI breaks the importance of a node into three metrics:

- The normalized importance of a node to the application;
- The location of interest under a specific system state: in the instance of fire detection, every ceiling sensor in a building may be of equal importance but the nodes monitoring a safe may become critical to the application after a fire is detected;
- The variables of interest under the previous two parameters.

SRI is the evaluation of sensors based on the weighted combination of these metrics. SRI helps weed out high importance nodes from a pool of possible routers.

Simulations show that SRI more than doubles the network lifetime under a fire alarm application by keeping important sensors alive for critical (raging fire) states.

6.6 Summary

We have shown the advantages of using a middleware to supervise a cross-layer protocol: the combination of MiLAN and DAPR is able to continuously adapt the sensor network to the application QoS requirements. Gains in network lifetime and QoS are substantial when the system design spares sensors whose data are critical to the application. We have also shown the advantages of using MiLAN to adapt the QoS requirements over a design with non-adaptive QoS. Finally, MiLAN takes full advantage of

DAPR's flexibility by adjusting its tunable parameters such as complex queries, query interval, and application cost. Through this tight cooperation, MiLAN is able to effectively manage the underlying DAPR protocol while providing a level of abstraction to the application programmer.

This chapter has shown that middleware can help protocols dynamically adapt their behavior to application requirements, which illustrates some of the optimizations made possible by the Middleware Interpreter presented in Chapter 5.

With the wealth of information from X-Lisa, the Middleware Interpreter and middleware now available at all levels of the protocol stack, new advances can be planned for the future of sensor networks. Next, we investigate such advances that enable adaptation of the protocol for the family of channel probing MAC protocols.

Chapter 7

Schedule Adaptation of the Low-Power-Listening MAC Protocol Family

Applications for wireless sensor networks (*WSN*) are becoming increasingly complex, and they require the network to maintain a satisfactory level of operation for extended periods of time. Consequently, sensor networks have to make the best possible use of their initial energy resources, specifically by constantly adapting their protocols to the changing conditions in the network. Both protocol-specific and cross-layer schemes have offered a plethora of energy reducing techniques. In particular, there are several protocols that focus on reducing energy at the data link / MAC layer, which constitutes the scope of this work. In this chapter, we investigate how to keep the radio in its energy-conserving sleep mode for as long as possible.

We offer three ways to adapt several key aspects of MAC protocols. The first idea, presented in this chapter, discusses switching between MAC schedules to adopt the most energy-efficient pattern of packet transmissions and receptions. Because different areas in the network experience different and changing loads of traffic, the MAC protocol should utilize the schedule most economical for the local conditions. Secondly, we propose to synchronize nodes in Chapter 8 so as to reduce transmission time and thus energy consumption and packet delivery delays. A third technique controls the inter-listening time to conditions in the network and is exposed in Chapter 9.

7.1 Introduction

As new sensor platforms have appeared on the market, a simple observation was made that idle listening, far from being negligible, was a major source of energy consumption [18] [19] [20]. Low-Power-Listening (*LPL*) and Preamble Sampling (PS) MAC protocols were introduced as a result. In his taxonomy of MAC protocols [15], Langendoën identifies *LPL* and *PS* protocols as two branches of random access MAC protocols, with the only difference that *LPL* MAC protocols need not know anything about their neighbors and their wake-up schedules. Both types of MAC protocols, including B-MAC [18], WiseMAC [17], SyncWUF [77] and X-MAC [21], use the insight behind Aloha with PS [16]: the sending node occupies the medium for long t_i[1] intervals to signal its imminent packet transmission. Receiving nodes are thus allowed to sleep for at most the duration of this preamble (t_i s), and they must stay awake when they sense a busy medium until the packet transfer is complete. In this work, we consider only the *LPL* branch of the Langendoën taxonomy (although many of our results can be transposed to other MAC protocols), and we define "(*LPL*) MAC schedule" as the pattern of packet transmissions occurring within the t_i interval.

Changes in radios have forced researchers to abandon B-MAC and a few other *LPL* protocols in some cases: although it paved the way to new MAC protocols, B-MAC, which uses a variable-length preamble to signal the impending packet transmission, can no longer be implemented as proposed on the new IEEE 802.15.4 compliant platforms because the IEEE 802.15.4 standard has a fixed preamble length of only a few bytes. We assume such a target radio, and make design and research decisions accordingly—thus B-MAC is not included in our work. After the introduction of new radios, researchers introduced new *LPL* protocols: X-MAC [21], C-MAC [23], and SpeckMac-D [22] are among the most popular contributions. These protocols are based on repeating either the data packet itself (SpeckMAC and CSMA-MPS), or an advertisement packet (X-MAC / C-MAC), in place of a long preamble. The details of the transmission schedules (the "MAC schedules") are given in Figure 7.1.

We prove that while the *LPL* family of MAC protocols generally lowers energy consumption without resorting to explicit exchange of active / inactive schedules between nodes, low duty cycles (or equivalently, high t_i values) drastically favor receiving nodes

---

[1] The notation t_i was borrowed from the work in [18].

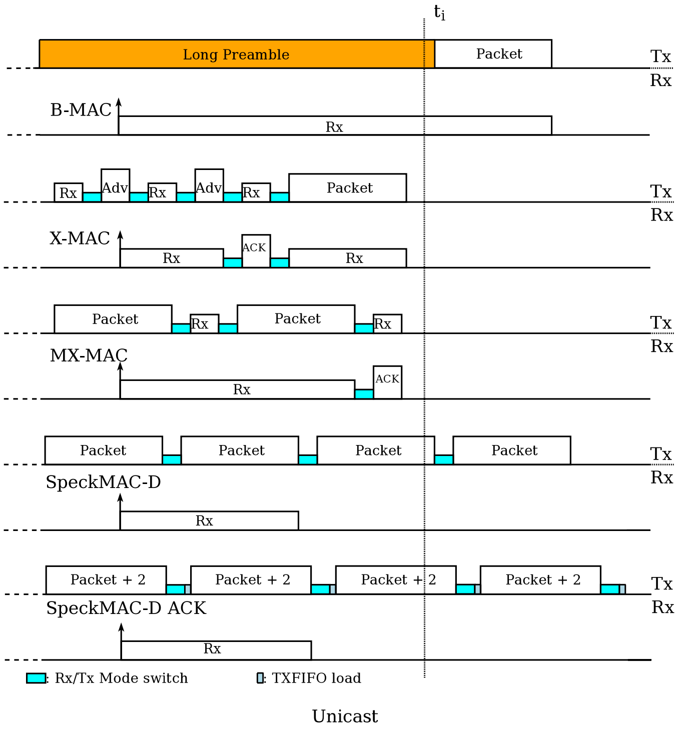


Figure 7.1: MAC schedule for B-MAC, X-MAC, MX-MAC, and SpeckMAC-D.

over mostly-sending nodes and induce higher delays and contention. As Figure 7.1 shows, for these *LPL* protocols, only one data packet can be transmitted per t_i cycle, which can cause a packet to experience high delay over several hops, and the network to deliver small data rates. Concern for delay may force network designers to select a high duty cycle that would limit energy savings. We address this problem in Chapter 8 by synchronizing the transmitting / receiving schedules of nodes on a slowly-changing routing tree.

This chapter's contributions are twofold:

- We propose switching MAC schedules from a pool of MAC protocols at the transmitter to minimize energy consumption based on parameters such as packet size, whether the packet is broadcast or unicast, and the estimated ratio of transmit to receive packets in the local neighborhood. The protocols are "compatible" because they are interchangeable: the receiver does not need to know what specific schedule is being used, it simply wakes up and senses the channel every t_i seconds and sends an ACK frame when required by the received packet. As a consequence, this protocol, called *MiX-MAC*, requires no overhead, and our implementation of this approach shows that lifetime gains can reach up to 30%.

- Because we utilize existing MAC protocols for our pool, we provide a detailed study of two existing *LPL* MAC protocols, X-MAC and SpeckMAC, in a head-to-head comparison, showing the advantages and disadvantages of each approach for both unicast and broadcast packets. We also identify MX-MAC, a modified version of CSMA-MPS, that is compatible with the X-MAC and SpeckMAC schedules.

7.2 MiX-MAC: A Highly Adaptable MAC Protocol

7.2.1 Principles of MiX-MAC

No protocol in the *LPL* family outperforms the others over all potential conditions in the network. Selecting a MAC protocol supposes a compromise between excellent performance under certain circumstances (hoped to be the common case), and suboptimal operation otherwise. Various protocols may perform differently according to the broadcast / unicast nature of the exchanged packets, the size of the packets, or whether a node is mostly receiving or sending packets. Adapting the MAC schedule allows optimal performance across those parameters.

Additionally, adaptation must occur during runtime since the traffic patterns in the network may not be known *a-priori*. In a tracking application for instance, the appearance of an object modifies the ratio of broadcast-to-unicast packets (more unicast packets are sent by nodes located in the neighborhood of the stimulus) and their sizes (reports on the object can be large). A MAC protocol chosen for its good performance when no object is detected would probably be sub-optimal when a target is being tracked.

We propose creating a pool of MAC schedules that are *compatible* with one another: while the sender may decide which schedule to follow based on the parameters mentioned above, the receiver need not be informed of the changes in MAC schedules. For instance, a sender choosing a certain MAC schedule may expect an ACK frame between packet transmissions; it will thus stay in receiving mode for a given time before it returns to transmitting mode. At the other end of the communication, a receiver simply wakes up periodically, and occasionally receives packets. If a received packet is marked with an acknowledgment request, it immediately sends an ACK frame. Switching between interchangeable MAC schedules guarantees that gains in energy and latency are achieved without any overhead other than the computation required to determine the best schedule to use. We call this approach, whereby the MAC schedule is adapted over time, MiX-MAC.

A small look up table within MiX-MAC helps in deciding what schedule is best suited for the current node, network and application conditions. This solution proves inexpensive in terms of computation power during runtime. The threshold values dictating a change in the MiX-MAC schedule can be established before deployment using simulation and implementation results. As we will show, inaccuracies in the estimates of current node, network and application conditions do not have a major impact on the performance of MiX-MAC.

Existing MAC protocols were included as part of the pool of compatible MAC schedules: X-MAC [21] and SpeckMAC-D [22], which were introduced around the same time. However, we also added an adaptation of CSMA-MPS, called MX-MAC.

7.2.2 X-MAC: A Short Preamble MAC Protocol

Under the X-MAC [21] schedule, a sender repeats the transmission of an advertisement packet containing the address of the intended receiver. Upon hearing the advertisement packet, the receiver replies with an ACK, which is followed by the transmission of the data packet by the sender. Figure 7.1 illustrates this process.

In [21], Buettner et al. do not propose implementing X-MAC for broadcast packets. X-MAC cannot broadcast packets as is, as the flow of advertisement packets cannot be answered by an ACK packet. A natural extension to X-MAC is to repeat advertisement packets for t_i and then send the data packet; however, receiving nodes have to wait until the completion of the advertisement cycle before they can receive the data packet—and go back to sleep (on average, they must wait for $\frac{t_i}{2} + t_{txPacket}$). In such cases, X-MAC performs equally to B-MAC, with the added advantage that it can be implemented using fixed preambles.

7.2.3 MX-MAC: a *LPL* Variant of CSMA-MPS Compatible with X-MAC and SpeckMAC Schedules

In [21], Buettner et al. make a convincing case for the energy and latency gains achieved by their proposed X-MAC protocol. Although efficient for unicast packets, this simple scheme is not well suited for broadcast transmissions. One additional drawback to X-MAC is its sensitivity to the hidden node problem and the persistence of a high risk of false positive packet reception acknowledgements. Indeed, early ACKs are sent and received before the data packet is transmitted, which does not guarantee successful reception of the packet.

Although it was introduced prior to X-MAC, CSMA-MPS [78] can be seen as a modification of X-MAC suitable for broadcast transmissions. CSMA-MPS repeats the data packet with its own wake-up schedule information and waits for ACK frames between transmissions. A received ACK signifies that the data packet has been correctly received and stops the transmission flow of data packets. This renders the MAC protocol immune to false positive packet receptions.

Although Mahlknecht et al. do not mention this point directly, the MAC schedule they propose can be adapted to broadcast packet transmissions so long as the sender does not request acknowledgment of the frames. Consequently, multiple receivers of the same packet may wake up, stay in RX mode until the full reception of a packet, and go back to sleep.

However, because CSMA-MPS must include scheduling information in every frame, which we do not need thanks to the implicit synchronization presented in Chapter 8, and because it decouples channel probes from transmissions (much like WiseMAC, but unlike X-MAC), the MAC schedule presented by Mahlknecht et al. is not fully compatible with X-MAC and SpeckMAC. This is the reason why Langendöen [15] classifies it on the *PS* branch of MAC protocols. Therefore, we introduce MX-MAC, the *LPL* pendant to CSMA-MPS. Figure 7.1 illustrates the timeline for MX-MAC. In MX-MAC, the data packets contain no scheduling information, and a node may wake up only once per t_i period to probe the medium and possibly send a packet immediately following the probe.

7.2.4 SpeckMAC-D: Repeating the Data Packet

Another medium sensing protocol is SpeckMAC-D [22]. In SpeckMAC-D, if a sender wants to transmit a packet to a receiver, it performs a clear channel assessment (CCA), and if successful, starts repeating the packet for at least t_i s. When a receiver wakes up, it checks the medium. If busy, it listens until it has received a full data packet or until it realizes that it is not the intended destination for the packet.

Figure 7.1 illustrates the transmission schedule for SpeckMAC-D.

Although not specified by Wong and Arvind in [22], we can imagine that ACKs be manually sent. If the sender is requesting an ACK, it needs to number the repeated data packets in a way that the receiver knows when to wake up and send an ACK after the sender is done transmitting. However, this small change adds two additional bytes of overhead in each packet in the absence of time synchronization; we consider that for t_i ranging from 0 to 1 s, 2 bytes are necessary to number packets (it is not uncommon to schedule more than 255 packet repeats for the smallest packet size and thus 1 byte would not be sufficient). This number has to be updated every time a packet repeat is sent out, forcing the MAC protocol to reload the TXFIFO buffer before switching to Tx mode. Figure 7.1 illustrates the transmission schedule for this protocol, labeled SpeckMAC-D-ACK.

7.2.5 Lifetime Calculation

MiX-MAC alternatively uses the MAC schedules of X-MAC / C-MAC, MX-MAC or SpeckMAC-D based on a look-up table. In order to implement MiX-MAC, we must populate the table and find the appropriate switching thresholds through simulations and actual implementation of various scenarios, as discussed next.

Following the work of Polastre et al. [18], we provide a quick reference for calculation of X-MAC and SpeckMAC-D lifetimes in Table 7.2. The notations and values of the variables for the CC2420 802.15.4 radio used for these equations appear in Table 7.1.

For a given scenario, the lifetime can be calculated for known battery capacities by evaluating the total energy for average wake up times at the receiver. In the sequel, r is the frequency of data reports sent, and n is the number of neighbors communicating

Table 7.1: Notations and values for the CC2420 radio.

Operation	Time		Current	
	Val. (μs)	Notation	Val. (mA)	Notation
Init radio volt. reg.	300	t_{reg_on}	0.02	c_{reg_on}
crystal osc.	860	t_{osc_on}	negl.	c_{osc_on}
Turn Tx on	192	$t_{idle/tx}$	17.4	$c_{idle/tx}$
Turn Rx on	192	$t_{idle/rx}$	19.7	$c_{idle/rx}$
Switch Tx/Rx	192	$t_{tx/rx}$	19.7	$c_{tx/rx}$
Switch Rx/Tx	192	$t_{tx/rx}$	17.4	$c_{tx/rx}$
Clear Chan. Assessment	128	t_{CCA}	19.7	c_{CCA}
Tx 1 byte	32	t_{txb}	17.4	c_{txb}
Rx 1 byte	32	t_{rxb}	19.7	c_{rxb}
Sample sensor	1,100	t_{sensor}	20	c_{data}

with the node.

$WakeUpTime$ is the time at which a receiver wakes up and is uniformly distributed between 0 and t_i seconds. Eqn. 1 represents the number of advertisement packets the sender has to transmit within t_i and before the receiver wakes up for X-MAC. Eqn. 4 provides the number of packet repeats needed for SpeckMAC. Eqn. 2 specifies the amount of time spent in X-MAC sending advertisements and listening for ACKs—this time is then translated in terms of energy in Eqn. 3. Eqns. 5 and 6 are the equivalent of Eqns. 2 and 3 for SpeckMAC. Eqns. 7 and 9 evaluate the time spent receiving a packet. t_{rxAdv} ($t_{rxPacket}$) is the time to receive the remainder of an ongoing advertisement (packet) and the one directly after, and is a function of $WakeUpTime$. Eqns. 11 and 13 specify the time spent probing the medium and include turning the oscillator on (from the radio state "power down") and sampling the medium twice to compensate for potentially waking up between two packet transmissions (X-MAC) or once for $192\ \mu s$ (SpeckMAC-D). Naturally, the time for which a node sleeps is when it does not do anything else (Eqns. 17 and 18).

Table 7.2: Time and energy analysis of X-MAC (X) and SpeckMAC-D (S) for unicast transmissions.

Op		Analytical form	#
Tx	X	$m =$ function of wake up time and t_i	1
		$t_{tx} = r \cdot [m \cdot (t_{rx/tx} + L_{adv} \cdot t_{txb} + t_{tx/rx} + L_{ACK} \cdot t_{rxb}) + t_{rx/tx} + L_{packet} \cdot t_{txb}]$	2
		$E_{tx} = Vr \cdot [m \cdot (t_{rx/tx} \cdot c_{rx/tx} + L_{adv} \cdot t_{txb} \cdot c_txb + t_{tx/rx} \cdot c_{tx/rx} + L_{ACK} \cdot t_{rxb} \cdot c_{rxb}) + t_{rx/tx} \cdot c_{rx/tx} + L_{packet} \cdot t_{txb} \cdot c_{txb}]$	3
	S	$m =$ function of t_i	4
		$t_{tx} = r(m+1)(t_{rx/tx} + L_{packet} \cdot t_{txb})$	5
		$E_{tx} = Vr(m+1)(t_{rx/tx} \cdot c_{rx/tx} + L_{packet} \cdot t_{txb} \cdot c_{txb})$	6
Rx	X	$t_{rx} = nr[t_{rxAdv}(wakeUpTime) + t_{rx/tx} + L_{ACK} \cdot t_{txb} + t_{tx/rx} + L_{packet} \cdot t_{txb}]$	7
		$E_{rx} = Vnr[(t_{rxAdv} + L_{packet} \cdot t_{txb}) \cdot c_{rxb} + t_{rx/tx} \cdot c_{rx/tx} + L_{ACK} \cdot t_{txb} \cdot c_{txb} + t_{tx/rx} \cdot c_{tx/rx}]$	8
	S	$t_{rx} = nr[t_{rxPacket}(wakeUpTime)]$	9
		$E_{rx} = Vnr \cdot t_{rxPacket} \cdot t_{rxb} \cdot c_{rxb}$	10
Probe	X	$t_{probe} = [t_{osc\_on} + 2(t_{idle/rx} + t_{CCA}) + t_{idle}]\frac{1}{t_i}$	11
		$E_{probe} = [t_{osc\_on} \cdot c_{osc\_on} + 2(t_{idle/rx} \cdot c_{idle/rx} + t_{CCA} \cdot c_{rxb}) + t_{idle} \cdot c_{idle}]\frac{V}{t_i}$	12
	S	$t_{probe} = (t_{osc\_on} + t_{idle/rx} + 2t_{CCA})\frac{1}{t_i}$	13
		$E_{probe} = (t_{osc\_on} \cdot c_{osc\_on} + t_{idle/rx} \cdot c_{idle/rx} + 2t_{CCA} \cdot c_{rxb})\frac{V}{t_i}$	14
Sense	X/S	$t_{data} = r \cdot t_{sensor}$	15
		$E_{data} = V \cdot t_{data} \cdot c_{data}$	16
Sleep	X/S	$t_{sleep} = 1 - t_{tx} - t_{rx} - t_{probe} - t_{data}$	17
		$E_{sleep} = V \cdot t_{sleep} \cdot c_{sleep}$	18

7.3 Fine Tuning Channel Probing Protocols

In this section, we describe some possible improvements that can be brought to channel probing protocols, specifically the X-MAC family and SpeckMAC-D.

7.3.1 Adapting the Channel Probing Interval t_i

7.3.1.1 Lifetime considerations

The channel probing interval t_i has a different impact depending on whether a node is sending or receiving packets. It can be seen as both an estimate of the time spent sleeping between two channel probes and the time spent sending packet repeats. Although usually fixed, t_i may take different values in different regions of the network. Obviously, in such cases, two nodes communicating with each other need to share their value of t_i. A node may only transmit with a value of t_i greater than (wasteful) or equal to that of the destination in order to ensure the receiving node will wake up during the transmission.

Although simpler, a common t_i throughout the network may not be appropriate in all cases. Variations in sensor node densities throughout the network may make uniform t_i values impractical.

Longer t_i values favor receiving nodes since a packet is almost always available (in both X-MAC and SpeckMAC-D) when they wake up. On the other hand, a longer t_i has the opposite effect in sending nodes since more packet repeats have to be sent (for a fixed packet size). While the number of neighbors (and their cumulative energy) is a good indication of a node's importance to the network as a whole, it is not a sufficient parameter to determine t_i because of adverse effects on sending and receiving nodes' lifetimes.

Optimal t_i can be obtained through simulation or analytically as suggested in [21] for certain send-to-receive ratios. As the average send-to-receive ratio changes, so should t_i.

In Figure 7.2, we show the lifetime of the X-MAC and SpeckMAC-D protocols as a function of t_i and for various ratios of transmissions over receptions. These analytic results are obtained using the equations of table 7.2 and the values in table 7.1. As the number of transmissions increases, the optimal t_i value decreases from $500\ ms$ to $100\ ms$ (X-MAC) and $350\ ms$ to $50\ ms$ (SpeckMAC-D). Compared to X-MAC, the SpeckMAC-D curve is steeper as t_i moves away from its optimal point because many more (or many less) packets have to be transmitted by the sender. This phenomenon is exacerbated when more transmissions happen (*i.e.*, for higher transmissions over receptions ratios).

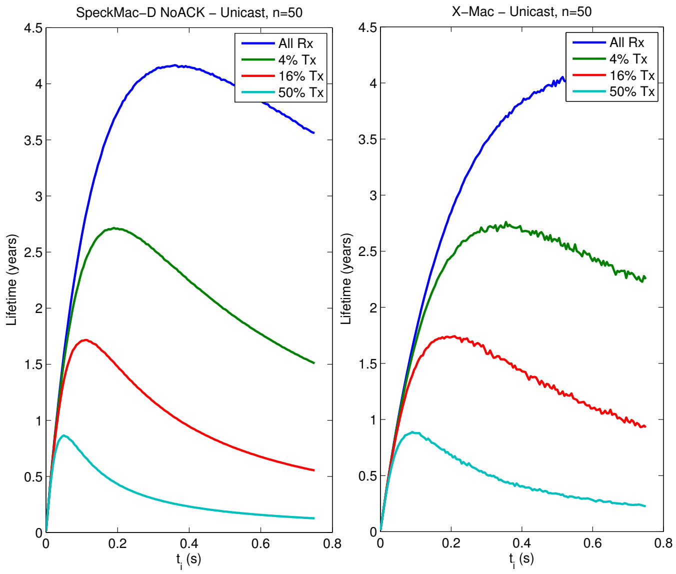


Figure 7.2: Lifetime as a function of t_i for SpeckMAC-D and X-MAC for various ratios of transmissions vs. receptions.

7.3.1.2 Contention considerations

In the previous subsection we observed that modifying the value of t_i had sometimes opposing effects on a node's lifetime. A protocol designer should also consider contention problems in the network. Contention in an area has adverse impacts on nodes that wish to send packets. When contention occurs, nodes tend to experience longer delays before they can transmit their packets, and thus waste energy probing the channel until it is free. Consequently, packets may arrive at their destination already stale, or may be dropped, causing costly packet retransmissions. Contention thus naturally affects lifetime.

An increase in t_i causes destination nodes to sleep for longer periods of time, and thus the average channel usage for one packet transmission increases linearly with t_i in low contention scenarios. Contention can be eased using various techniques:

- The application layer may choose to report only critical events to the base station.
- The routing protocol may elect routes that are not experiencing contention or poor link reliability.
- The MAC protocol may choose to lower the value of t_i.

- The physical layer may change the transmission power: decrease it to lower interference to other nodes, or increase it to reach contention-free nodes.

7.3.2 Adapting the Packet Transmission Schedule

All packets do not have the same importance for the network or for the application. For instance, a critical event detection packet is essential to the application and the end user, but carries very little significance for the life of the network. Conversely, route set-up packets derived from energy costs help increase the network lifetime but bear little meaning to the application.

While other protocols higher in the stack may reorder or suppress outgoing packets, the focus of this work is the MAC / data link layer. We propose integrating the following rules to any channel probing MAC protocol:

- Packets marked as urgent should be handled immediately, while other packets may be sent only at the scheduled wake-up time (that is, after t_i s of sleep). This feature allows longer lifetimes in nodes whose burden is mainly to transmit packets and who do not experience severe contention.

- Service packets, even though they may not be urgent, should precede non-urgent data packets. This should help provide the network with up-to-date information, thus avoiding route failures or node activation errors, among others.

- If a packet is part of a burst (*e.g.*, a succession of frames in a video application), the node should not go to sleep and should transmit subsequent packets immediately. Such features can be implemented easily under the 802.15.4 standard by setting the *frame pending* subfield (part of the *Frame Control Field*) to 1 after the completion of a packet (an ACK in some cases). The new frame is transmitted after a short random wait for fairness.

We illustrate a timeline of these simple rules in Figure 7.3. Each packet is designated by an ID number, its unicast (U) or broadcast (B) nature, its urgency (1 or 0), the nature of its contents, and the number of packets in the burst. Figure 7.3 shows the scheduling decisions taken by the MAC protocol based on information about the packets. Urgent packets are treated immediately (packet 2), and service packets have precedence over

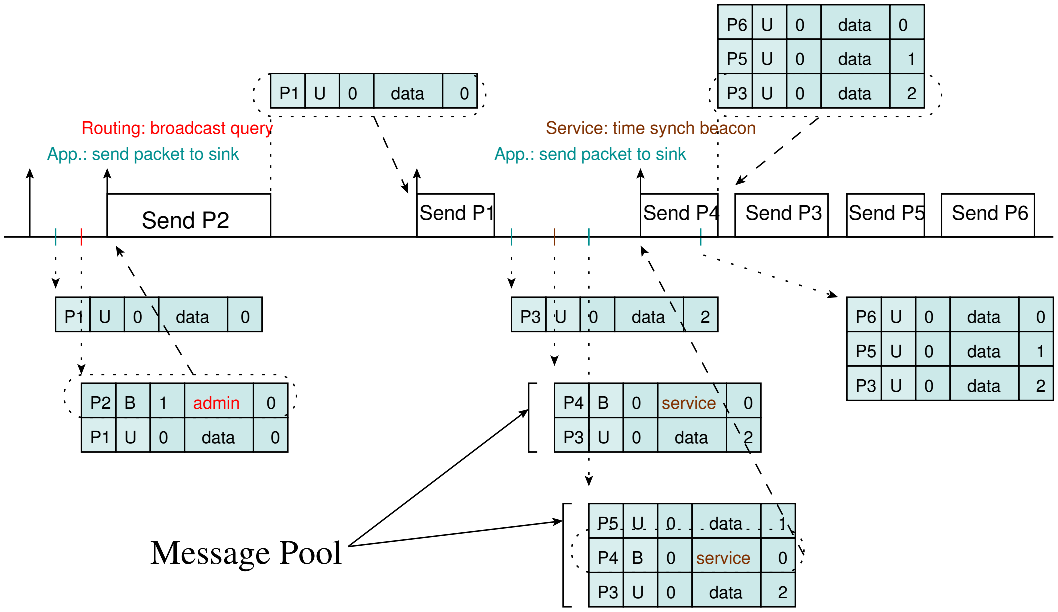


Figure 7.3: Adapting the node's schedule to packet types.

non-urgent data packets—data packet urgency is decided upon by the application requirements.

7.4 Simulation Comparison of MAC Protocol Scheduling Techniques

This section provides a direct comparison of the *LPL* MAC protocols X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK for both broadcast and unicast packets and for different ratios of transmit to receive and different packet sizes, in order to determine which MAC schedule performs best under different conditions.

The analytical model described by the equations in Table 7.2 can be seen as an "ideal" representation of the radio, when not burdened by an operating system. Using this model provides insights into the inner qualities and drawbacks of the various MAC protocols. The values for the parameters are selected based on the CC2420 radio. Section 7.5 provides a discussion of the implementation of these MAC protocols for TinyOS platforms.

Table 7.3: CC2420 Radio Parameters

Notation	Parameter	Value	Unit
C_{batt}	Capacity of battery	18000	As
V	Voltage	3	V
L_{packet}	Packet length	40	Bytes
L_{ACK}	ACK length	11	Bytes
L_{adv}	Advertisement length	11	Bytes
r	Packet rate	$\frac{1}{300}$	packet.s^{-1}

7.4.1 Performance Comparison

We begin by comparing X-MAC, MX-MAC, SpeckMAC-D and SpecMac-D-ACK for a scenario similar to the one described in [18]. One node receives packets at a rate r from n neighbors. This node sends m packets at rate r to one (unicast) or all (broadcast) neighboring nodes. neighboring node. Unless otherwise specified, m is equal to 1. The lifetime is calculated based on the values in table 7.3, and for the highest transmit power setting ($0\ dBm$). Note that the value for L_{adv} is set to 11 bytes which includes 5 bytes of preamble and 6 additional bytes for headers and footers as standardized by the CC2420 radio.

7.4.1.1 Broadcast packets

In this scenario, all packets are broadcast, so there are no ACK exchanges. However, SpeckMAC-D has to be divided in two categories: SpeckMAC-D-ACK (SMD-ACK) which adds two bytes of overhead to the packets and requires loading the TXFIFO before every transmission, and regular SpeckMAC-D (SMD), which does not. A programmer could not freely switch from one to the other because a fixed MAC header is required to access all the fields within the header. Since this decision has to be made prior to network deployment, we elected to compare both schemes.

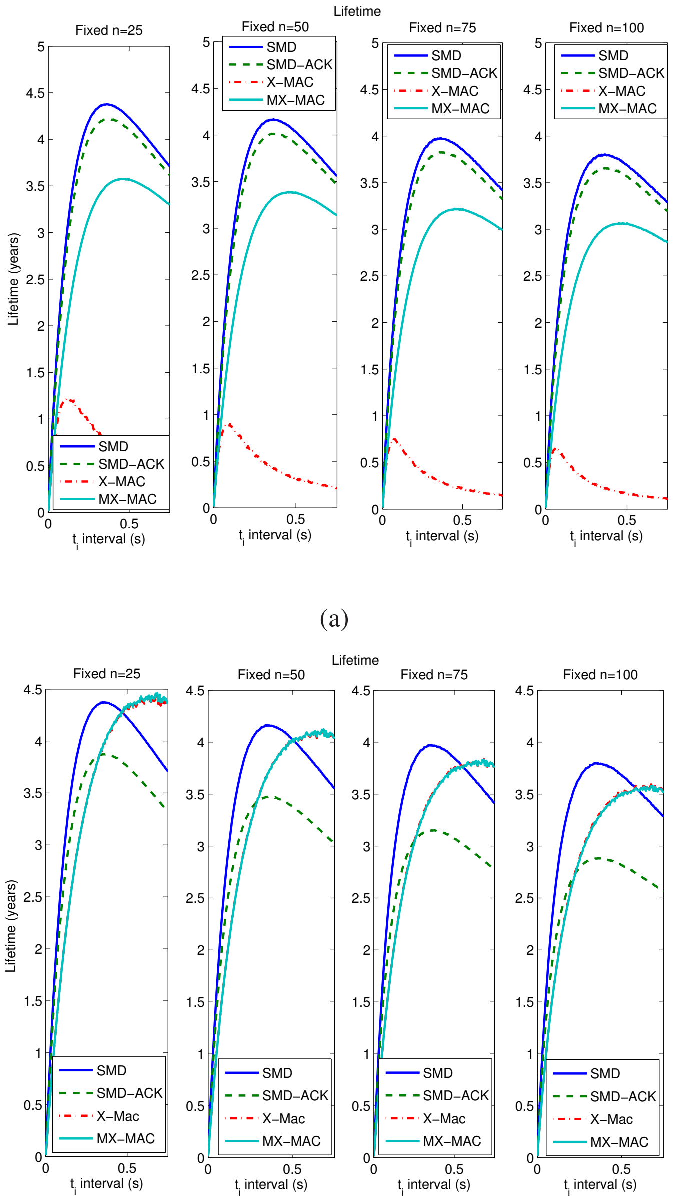
Figure 7.4(a) shows the lifetime given fixed t_i and n for X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK. As is to be expected, SpeckMAC-D performs better than SpeckMAC-D-ACK at all times and for all cases since the latter adds bytes of overhead to the transmission and keeps the radio active longer to reload the TXFIFO. MX-MAC and X-MAC both exhibit more modest lifetimes than SpeckMAC-D. X-MAC suffers from the problem exposed in Section 7.2.2: as t_i increases, so do the receive times at the nodes because in broadcast mode, the stream of advertisements cannot be interrupted. MX-MAC does not suffer from the same problem, but it still has a shorter lifetime than SpeckMAC-D, as MX-MAC forces the sending node to spend more time in Rx mode than the latter and Rx mode is more energy costly than Tx mode for CC2420 radios. The "optimal" t_i is approximately $400\ ms$ and $150\ ms$ for SpeckMAC-D / MX-MAC and X-MAC, respectively.

7.4.1.2 Unicast packets

In this scenario, all packets are unicast. Figure 7.4(b) shows the lifetime given fixed t_i and n for X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK. X-MAC and MX-MAC perform similarly, although the results show clearly the impact of packet size. In unicast scenarios, MX-MAC can be considered to be X-MAC with longer packets. When a receiving node running MX-MAC wakes up at a time when it misses the beginning of a new packet, it must stay in Rx mode longer than those running X-MAC. As n increases, this discrepancy materializes in greater proportion and causes the lifetimes of nodes running MX-MAC and X-MAC to cross-over. For both, the "optimal" t_i is in the interval $[500\ ms; 750\ ms]$. As m is 1, the main task for the studied node is to receive, causing the optimal t_i to be larger than in $m \approx n$ scenarios, and as explained in Section 7.3.1.1.

SpeckMAC-D gives a completely different picture of the evolution of its lifetime as a function of t_i. Most surprisingly, SpeckMAC-D, in spite of repeating data packets for at least $t_i\ s$ without interruption, brings a lifetime comparable to the X-MAC protocols. However, this is achieved for smaller values of t_i such that a sending node would spend much less time in Tx mode. Two observations should be made: the optimal value of t_i is a narrow "sweet spot" and SpeckMAC-D without ACKs puts the burden of acknowledgment on upper layers (not included here). SpeckMAC-D-ACK follows the same pattern, although its lifetime is reduced by increased size and time overhead. These results motivate several observations:

- If no ACK is required by the protocol issuing the packet, a longer lifetime may



(a)

(b)

Figure 7.4: Node's lifetime for X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK with a fixed n. All packets sent are (a) broadcast, (b) unicast.

be achieved with SpeckMAC-D provided that t_i can be kept around its optimal value and that contention is low (SpeckMAC-D utilizes the channel for the entire t_i s).

- Otherwise SpeckMAC-D-ACK may provide the longest lifetime under similar conditions although for a smaller domain. SpeckMAC-D-ACK spends less time in Rx mode (more energy consuming in the CC2420) than X-MAC. As t_i increases however, this advantage pales in the face of numerous packet repetitions.
- For unicast packets, the relative performances of X-MAC and MX-MAC are comparable but smaller neighborhood sizes are better suited to the latter, yielding minute lifetime increases.

7.4.1.3 Ratio of transmissions vs. receptions ($m \approx n$)

As explained in Section 7.3.1.1, a node's ratio of transmissions to receptions may shift the optimal value for t_i by a considerable amount. Figure 7.5 shows the changes in node lifetime when the node sends approximately as many unicast packets as it receives (the unlikely scenario where the node forwards all received *broadcast* packets is not shown).

X-MAC and MX-MAC still perform equally well, although a small difference can be perceived: the cross-over observed above no longer happens because nodes running X-MAC send many more packets, which, due to the X-MAC schedule, sets the radio in Rx mode for longer periods of time than for MX-MAC. Since advertisement packets are shorter than data packets, X-MAC spends more time switching modes and listening to the medium than MX-MAC. As expected, the optimal t_i value is significantly smaller than before.

For $m \approx n$, the optimal t_i sweet spot is much more narrow. Changes made to t_i by the MAC protocol must be very gradual because the effect of transmitting a packet vs. that of receiving a packet can be very severe outside of a very narrow band within which the MAC protocol must operate.

7.4.1.4 Packet size

Figure 7.6 shows interesting results when the packet size is doubled from 40 bytes to 80 bytes and unicast transmissions are used with $m = 1$. In this case, X-MAC's maximum lifetime is equal to that of SpeckMAC-D. As missing the beginning of a packet transmission requires staying in Rx mode for longer periods of time, X-MAC does better than MX-MAC.

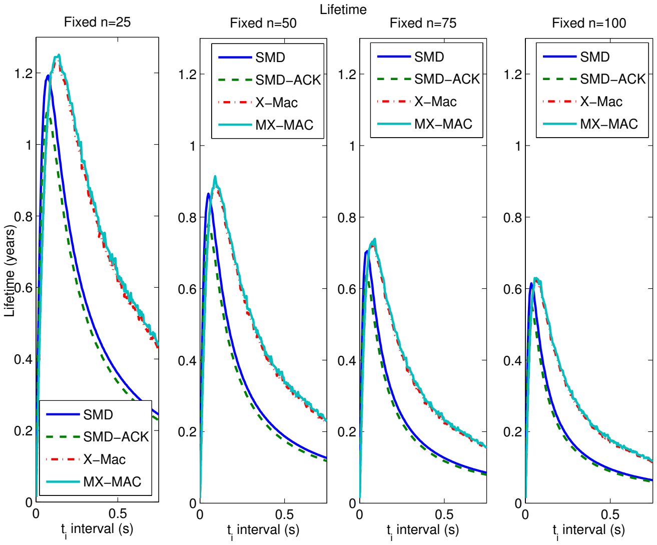


Figure 7.5: Node's lifetime for X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK with a fixed n. All packets sent are unicast, and $m = n + 1$.

For large packets, choosing X-MAC over MX-MAC can increase the lifetime of individual nodes by up to 30%. It could also be the motivating factor to abandon SpeckMAC-D for X-MAC, which is also less prone to contention. Consequently, packet size is a relevant parameter in choosing the best schedule for a packet transmission—and is easily obtained information.

7.4.1.5 Contention and Delays

In this section, we simulate the impact of each protocol design on contention and delays. The SpeckMAC-D protocol occupies the channel for the duration of $t_i\ s$ while the X-MAC and MX-MAC schedules do so for only ${}^{t_i}\!/_2\ s$ on average. The X-MAC and MX-MAC schedules allow packets transmissions to be "staggered" over t_i. SpeckMAC-D can complete only one transmission during that interval.

In order to simulate contention, we used a simple model where n nodes wish to send m packets at a rate r. The channel is considered busy if any transmission is ongoing. In our simulation, ten nodes have $150\ s$ to transmit ten packets at a rate ${}^{1}\!/_{5}$. We assume that the same event triggered all nodes to start transmitting a packet within five seconds.

Figures 7.7(a) and 7.7(b) show the packet delivery ratio and time for which the

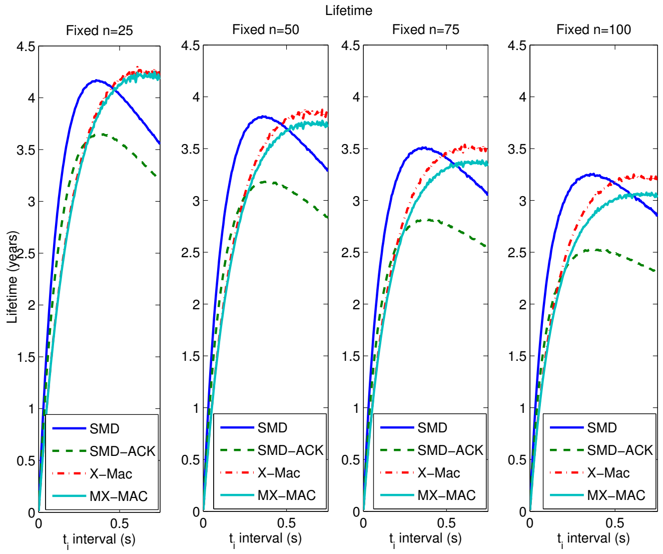
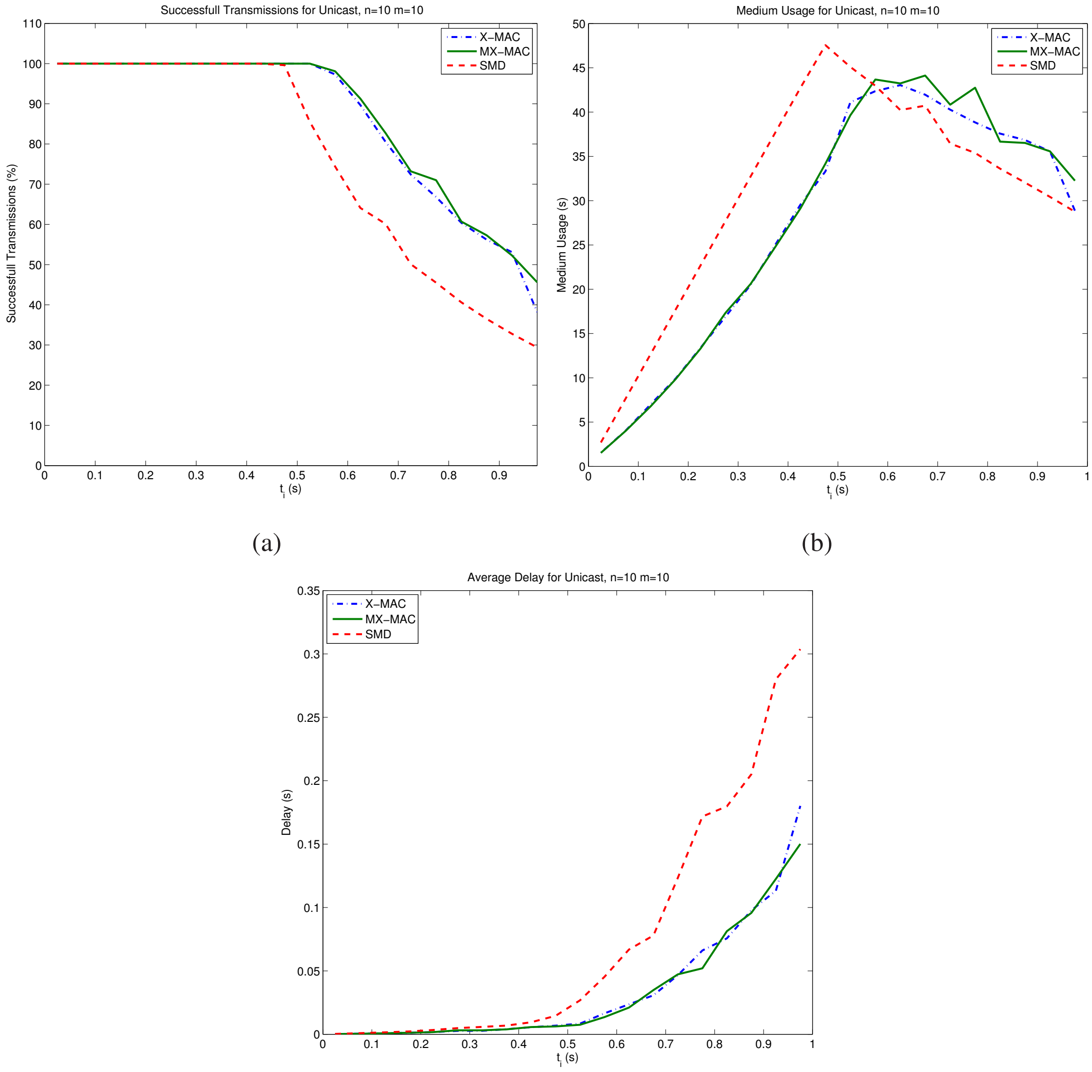


Figure 7.6: Node's lifetime for X-MAC, MX-MAC, SpeckMAC-D and SpeckMAC-D-ACK with a fixed n. All packets sent are unicast, and the packet size has doubled to 80 bytes.

medium is busy for all three protocols. When t_i increases, the channel is more often busy for all protocols, but more so when the SpeckMAC-D schedule is used. In Figure 7.7(b), we can see that X-MAC and MX-MAC only occupy the channel for approximately half the time for SpeckMAC-D until t_i exceeds a threshold value after which packets are dropped. This threshold is smaller for SpeckMAC-D than for the X-MAC and MX-MAC schedules. The direct consequence is to increase the delay as shown by Figure 7.7(c). SpeckMAC-D becomes more sensitive to delay as t_i increases, although it remains relatively small ($300\ ms$) in this however very demanding scenario. On average, the delay incurred by SpeckMAC-D is twice that of the other two protocols.



(a)

(b)

(c)

Figure 7.7: 5 nodes attempt to send 10 packets at a rate $1/5$ for various t_i values. (a) gives the packet delivery ratio, (b) the channel usage, and (c) the average delay.

7.4.2 MiX-MAC: Adapting the MAC Schedule to Conditions in the Network

7.4.2.1 Picking the right MAC schedule

The previous section shows that all MAC protocols sacrifice performance in unicast mode to that of the broadcast mode or vice-versa. We show that MiX-MAC performs well against every combination of parameters.

SpeckMAC-D-ACK does not belong to the set of compatible protocols since placing a 2 byte number sequence within packets requires universal *a-priori* knowledge of the MAC protocol to be used, and this must remain fixed during runtime. B-MAC, however, would be part of the pool if it could be implemented on the CC2420 radio.

MiX-MAC adopts SpeckMAC-D's schedule for broadcast packets, and for unicast packets, it uses four axes to decide the appropriate schedule. These include t_i value, packet size, estimated ratio of transmitted vs. received packets, and the ACK requirements determined by the upper level protocols or services.

A small look up table within MiX-MAC helps in deciding what schedule is best suited for the current node, network and application conditions. The threshold values dictating a change in the MiX-MAC schedule can be established before deployment using figures such as Figures 7.4(a) through 7.6.

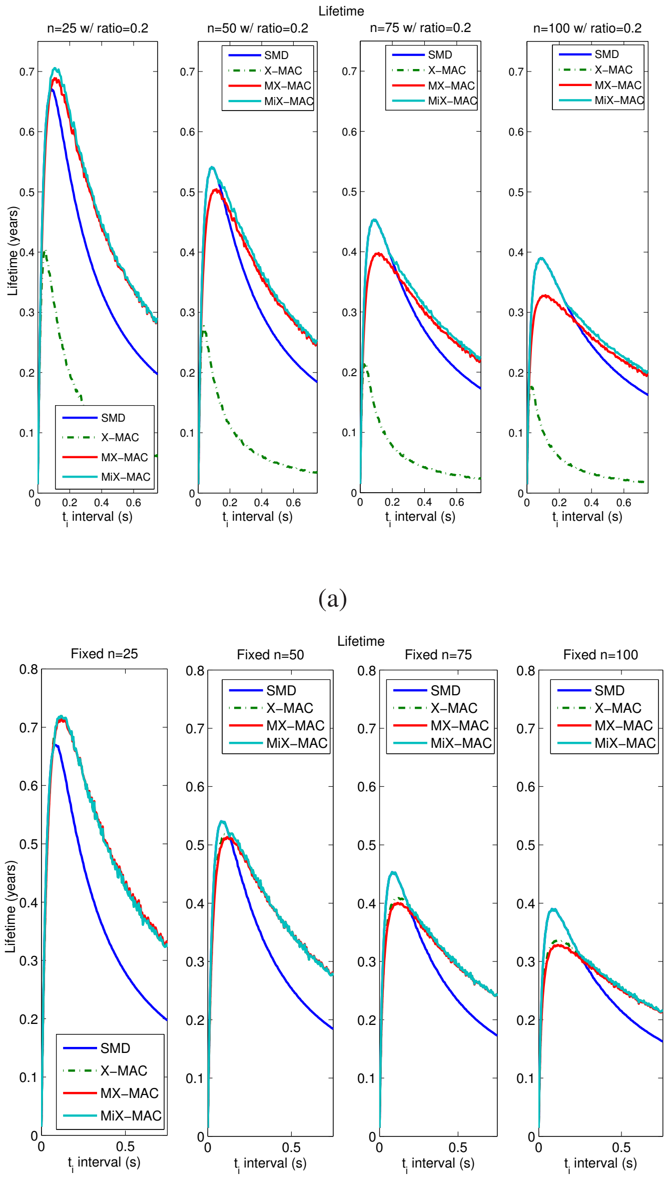
7.4.2.2 Resulting lifetime increase

Figure 7.8 presents a comparison of the lifetimes for MiX-MAC, X-MAC and SpeckMAC-D for a packet size of 40 bytes, 20% of the total traffic being broadcast packets (all other packets are unicast) (Figure 7.8(a)) and all unicast packets (Figure 7.8(b)). The rate r is $^1/_{10}$.

These results show that MiX-MAC achieves the upper bound of node lifetime by selecting the best schedule for various scenarios. X-MAC suffers greatly in broadcast mode, even for relatively small proportions of broadcast packets (20%). MiX-MAC helps the MAC protocol obtain the best of all worlds: lifetime gains are obtained over other protocols on a full range of t_i values. This last point is important because in a network where the rate of packet transmissions varies, the optimal value of t_i would naturally vary as was seen previously.

7.5 TinyOS Implementation of *LPL* Protocols

This section compares *LPL* MAC protocols implemented in TinyOS for the Tmote Sky platform in order to find the switching thresholds of the MiX-MAC look-up table.



(a)

(b)

Figure 7.8: Node lifetime as a function of t_i for MiX-MAC, X-MAC, MX-MAC and SpeckMAC-D. The packets are 20% broadcast / 80% unicast (a) and all unicast (b), sent at a rate $r = \frac{1}{10}$.

Because in the TinyOS implementation, the relatively slow CPU must operate sensors, protocols, and the CC2420 radio, our simulation model may need refining or amending. We also provide guidelines for rigorous MAC protocol evaluation, and we underline the strengths and weaknesses of more simple simulation models. We also offer a sense of which model assumptions are valid, and which do not hold.

7.5.1 Debugging at the MAC Level

TOSSIM is a platform emulator for PC's under TinyOS and does not model MAC protocols in details. Consequently, it can be very difficult to debug MAC protocols in TinyOS as the use of LEDs is very limiting and is not adapted to fast-paced operations as typically encountered when programming MAC protocols. Designing a debugger was a critical step in developing a MAC protocol. The easiest solution involves the UART, which allows the Tmote to communicate with the PC. The debugger consists of markers (whose significance can be defined by the programmer) that may be recorded to an array by invoking:

```
call Debugger.debug(value);
```

The array of debugging values is periodically sent to the PC through its USB connector for print out on the screen. Thus, it allows the programmer to follow what operations are sequentially being performed by the protocols.

7.5.2 Reconstruction Model

Since we are comparing four different MAC protocols along a host of parameters such as t_i values, number of received and sent packets (n and m), packet size, and the unicast or broadcast nature of the packet, it is impractical to measure the energy consumed by a mote over its lifetime as we vary all of these parameters. While possible, measuring the actual battery voltage over the life of the mote does not map linearly with energy consumption. Moreover, such a method would have forced us to wait well over six months for every created scenario. Instead, we chose to accurately evaluate the lifetime of a mote by measuring the current drawn under various basic operations using a fast data acquisition board. Figure 7.9 is a picture of the acquisition board plugged into the computer during a measurement.

Figure 7.10(a) provides a sample of the output from the data acquisition board when the mote is probing the medium. Figure 7.10(b) shows the current drawn by the mote

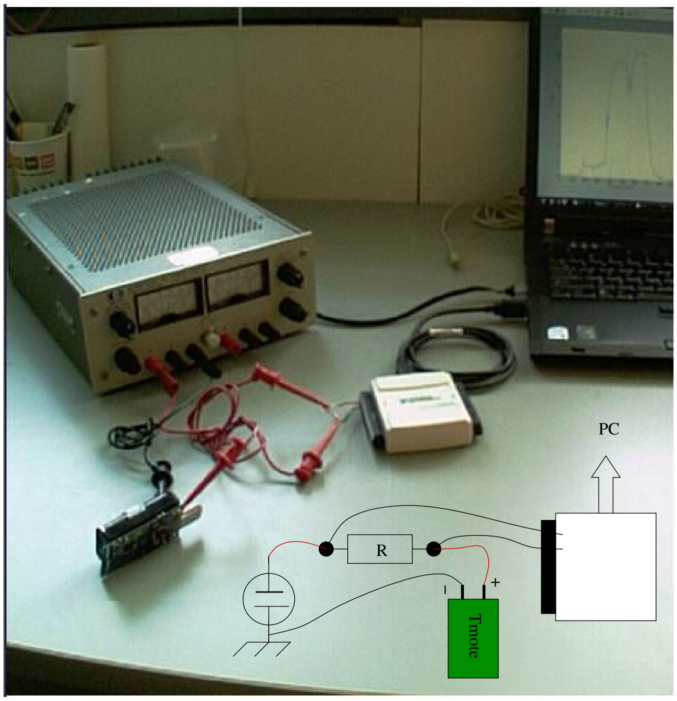
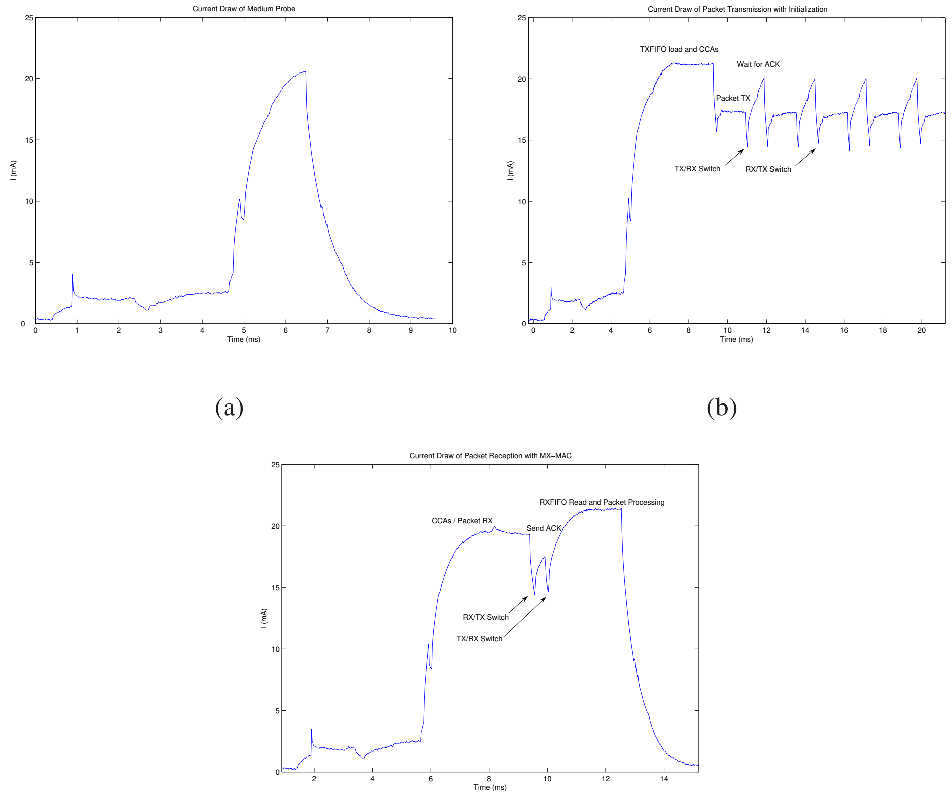


Figure 7.9: Picture of the acquisition board set up to measure the current drawn from the mote.

during a packet transmission initialization and the first five packet repetitions. Figure 7.10(c) shows the amperage for a packet reception, followed by an ACK. The identifiable radio operations are indicated on Figure 7.10 and account for most of the energy consumption. Since the measurements were done at a constant voltage, and the samples taken at a fixed and small rate, the energy can be easily calculated through Riemann Integrals.

We measured the energy and time spent probing the medium (*probe*), starting a transmission (*startTx*), sending one frame and switching the radio back to TX mode (*frame*), stoping a transmission after a successful (*endTxS*) and failed (*endTxF*, only for X-MAC and MX-MAC schedules) transmission, and receiving a packet (*Rx*). Some of these measurements were repeated for various packet sizes in order to find a correlation between energy expended and packet size. Linear regression was subsequently used to create a model for various radio operations, all with an R value greater than or equal to 0.99.

The $t_{frame}*$ value in table 7.4 verifies that our model correctly represents TinyOS



(a) (b)

(c)

Figure 7.10: Current drawn by the Tmote Sky during a medium probe (a), a $40\ B$ packet transmission (b), and a $40\ B$ packet reception (c).

and the CC2420 radio: all protocols show that it takes a little under $32\ \mu s$ to send one byte (we measured $31.89\ \mu s$ exactly), which is confirmed by the CC2420 datasheet. Moreover, our measurements allow us to determine the *turn around* time for each protocol. For SpeckMac based schedules, the time to revert to TX mode is $772\ \mu s$ and it is $1.351\ ms$ for MX-MAC and X-MAC based protocols. These latter values depend heavily on the TinyOS code and may differ from one programmer to the next.

In order to validate further our reconstruction model, we compared it to real-life measurements on the mote. Because Matlab, which is used to obtain the data values from the data acquisition board, is limited in vector sizes, we could only acquire ten minutes of current draw and had to break the acquisition in five samples of two minutes. We ran three scenarios on the mote: we used various values of m and n, as well

Table 7.4: Actual radio model under TinyOS. S designates the packet size in bytes. Units are μJ and ms.

<table>
<tr><th>Notation</th><th>Speck</th><th>MiX as. Speck</th><th>MX</th><th>X</th></tr>
<tr><td>E_{probe}
t_{probe}</td><td>132.4
8.9</td><td colspan="3">142.6
9.6</td></tr>
<tr><td>$E_{startTx}$
$t_{startTx}$</td><td>$1.1 \cdot S + 155.9$
$0.02 \cdot S + 6.7$</td><td>$1.3 \cdot S + 206.7$
$0.02 \cdot S + 7.5$</td><td>$1.2 \cdot PS + 230.3$
$0.02 \cdot S + 7.9$</td><td>264.8
8.4</td></tr>
<tr><td>E_{TxAdv}
t_{TxAdv}</td><td colspan="3">NA
NA</td><td>138.2
2.6</td></tr>
<tr><td>E_{frame}
$t_{frame}*$</td><td colspan="2">$1.7 \cdot S + 35.1$
$0.03 \cdot S + 0.8$</td><td>$1.7 \cdot S + 70.2$
$0.03 \cdot S + 1.4$</td><td>$1.7 \cdot S + 63.6$
$0.03 \cdot S + 1.2$</td></tr>
<tr><td rowspan="2">E_{endTxF}
t_{endTxF}
E_{endTxS}
t_{endTxS}</td><td colspan="2">71
3.7</td><td rowspan="2">75
3.8
144.3
4.9</td><td rowspan="2">72
3.5
144.3
3</td></tr>
<tr><td>NA
NA</td><td>NA
NA</td></tr>
<tr><td>$E_{Rx} = \mathcal{N}(\mu, \sigma)$</td><td>$\mu = 5.9 \cdot S + 166.7$
$\sigma = 43$</td><td>$\mu = 6 \cdot S + 163.6$
$\sigma = 80$</td><td>$\mu = 5.1 \cdot S + 208.2$
$\sigma = 70$</td><td>$\mu = 5.4 \cdot S + 488.6$
$\sigma = 66$</td></tr>
<tr><td rowspan="2">$t_{Rx} = \mathcal{N}(\mu, \sigma)$</td><td colspan="2">$\mu = 0.1 \cdot S + 9.3$</td><td rowspan="2">$\mu = 0.09 \cdot S + 10.4$
$\sigma = 1.2$</td><td rowspan="2">$\mu = 0.09 \cdot S + 14.8$
$\sigma = 1.1$</td></tr>
<tr><td>$\sigma = 0.76$</td><td>$\sigma = 1.1$</td></tr>
</table>

as different values of packet size. We sent packets using the SpeckMAC schedule to eliminate varying packet transmissions in order to fix the real-life scenario and match it to the reconstruction scenario. We also considered that random transmission durations, as found with MX-MAC and X-MAC, would not average out over a period of ten minutes, and thus, could not execute the desired scenario.

Figure 7.11 shows that the relative difference between the real-life scenario and its reconstruction does not exceed 3%. We have observed that most of the error emanates from the estimation of the idle power which tends to vary over time due to temperature changes in the acquisition circuit. In scenarios with little traffic (the star '$*$' scenario),

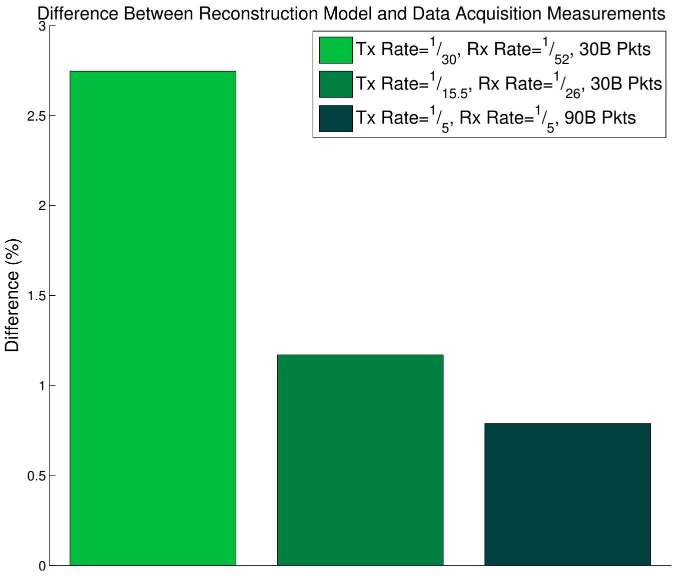


Figure 7.11: Relative difference between real life scenarios and their prediction through the reconstruction model.

the idle power accounts for a large part of the power consumption. As the amount of traffic increases, the relative share of the idle power decreases, and our model gains accuracy. When plugging the idle power value of the real-life scenario into our reconstruction model, the error dips under 1%.

7.5.3 Protocol Design Choices

We tried to optimize as many aspects of the MAC schedule as possible: the time separating two clear channel assessments (CCA) as well as the number of CCAs when sensing the medium, the number of CCAs before a packet transmission, the behavior of a node when detecting another ongoing transmission (a sender hearing another stream of packets during its switch to RX mode after every frame), etc. Since our goal is only to *compare* MAC protocols without bias toward one schedule, we endeavored to optimize the behavior of all three MAC protocols. Because all MAC schedules are meant to be compatible, they were implemented by the same TinyOS code. Consequently, all three protocols have the same essential parameters such as the number of CCAs and the time separation between them.

7.5.3.1 Time Separation Between CCAs

On CC2420 radios, the channel probing result is read on the CCA pin (Clear Channel Assessment) and can only be obtained after the radio has been in receive mode (RX mode) for at least 8 symbol periods (or $128\ \mu s$). The CC2420 data sheet indicates that switching from transmit (Tx) or idle modes to RX mode takes $192\ \mu s$. We assume that there is no formal time synchronization between nodes.

Two CCAs are usually sufficient to detect an ongoing transmission, provided they are separated by the correct time. We developed an analytical model to calculate the probability that a node would correctly hear an ongoing stream of packets as a function of the time between CCAs (t_{2CCAs}), the sender's *radio switch* time, and the size of the packets. Figures 7.12(a) and 7.12(b) plot the numerical value of this probability of successfully receiving the frame for the two radio switching times that we measured ($770\ \mu s$ and $1,350\ \mu s$). These figures show that good choices for t_{2CCAs} tend to be around $600\ \mu s$ and $1,100\ \mu s$. From the perspective of energy consumption, a shorter time separation between consecutive CCAs is beneficial because it shortens the time spent probing the medium.

Our Tmote Sky implementation helped us refine these values: because this model is an ideal representation of the radio, we coded values for t_{2CCAs} that were much smaller than $600\ \mu s$ and $1,100\ \mu s$ in order to account for the slower execution of the whole protocol stack on the Tmote Sky.

Figure 7.13 shows the packet delivery ratios for two nodes randomly sending 100 packets to each other, including collisions, missed packets, bad radio states, etc. As the packet size increases, it is generally easier for a receiver to hear a transmission. The dotted line shows a value for t_{2CCAs} that was not retained because of poor reliability.

We found that an acceptable coded value for t_{2CCAs} is between $320\ \mu s$ and $512\ \mu s$ for SpeckMAC, and $512\ \mu s$ for MX-MAC and X-MAC, which represent the best compromise between energy use in very low traffic networks and fairness to all protocols. For MiX-MAC, which must use compatible parameters for all MAC schedules, we set t_{2CCAs} to $512\ \mu s$ for all protocols.

For packet sizes close to their maximum value ($128\ B$), we found the radio to "jam" under SpeckMAC: the radio would issue RXFIFO overflows because the FIFO was filled before it could be read, and hence the packet delivery ratios in this case dropped significantly.

7.5.3.2 Automatic ACKnowledgment

For MX-MAC and X-MAC schedules, the automatic hardware ACKs were chosen. With the CC2420 radio, this means that automatic address recognition and automatic CRC check be enabled. The former allows the radio to signal it has heard the first bytes of a transmission to another node, to which TinyOS can answer with a power down command. Under these circumstances, we consider that overhearing is negligible compared to the overall medium probe and equally affects all protocols.

The automatic ACK is issued immediately after receiving a packet that is destined for the node, that passed the CRC check, and that requests an ACK. We chose automatic ACKs over triggered ACKs for their speed of execution which eventually led to a shorter t_{2CCAs}. Non-automatic hardware ACKs can only be triggered after reading the packet from the RXFIFO, which is typically a slow process.

In fact, the automatic hardware ACK mechanism is so rapid that assumptions about ACKs need to be rethought. In cases when the packet size is very large (above $80\ B$), the RXFIFO cannot be read before the ACK is sent. When no automatic ACK is enabled, the radio stays in receiving mode until the packet is read, whereas automatic ACK frames can be sent while the RXFIFO read-out is waiting completion. Since the TX mode consumes *less* energy than the RX mode on the CC2420, sending an ACK frame actually *reduces* the energy expended.

The use of automatic ACKs further motivated removing WiseMAC [17] from the list of compatible MAC schedules because each node must piggyback its wake-up schedule to acknowledgement frames. Moreover, doing so requires to use software acknowledgement, and to reload the TXFIFO (a time consuming process) every time. This results in a significant increase in energy consumption.

7.5.3.3 The SpeckMAC Design

Start of Frame Delimiter (SFD) captures notify the TinyOS code that frame delimiters have been received. This gives information about the state of current receptions. We suspended SFD capture notifications as soon as a packet is received in order to avoid overwhelming the TinyOS code with unnecessary interruptions. This is particularly needed in the case of SpeckMAC schedules because the flow of packets may not be interrupted, and constant irruption of time consuming events warning of the reception of a packet may delay their processing and the power down command of the radio. This, however, does not prevent the radio from receiving packets, which can be read out at a later time.

7.5.3.4 The X-MAC Design and Choice of Advertisement Packet Size

As Figure 7.13 shows, the packet delivery ratio for packets of size $11\ B$ is only 65% for X-MAC, even with t_{2CCAs} set to $512\ \mu s$. This prompted the choice of larger advertisement packets ($40\ B$), as is supposed by C-MAC [23]. In order to remain compatible with the other protocols, we considered all $40\ B$ long packets to be advertisements. The receiver, upon reading a $40\ B$ packet from its RXFIFO stays awake to receive the subsequent data packet. In other words, when a MAC protocol needs to send a $40\ B$ packet, it has to use the X-MAC schedule.

7.5.3.5 MAC Schedule Compatibility

Through design choices, we allowed the three MAC protocols to be compatible. This means that the same TinyOS code can let a mote send and receive packets using the MX-MAC, X-MAC or SpeckMAC schedule.

More importantly, the basic principle behind schedule compatibility is that a receiver does not need to know the ongoing schedule, and simply ACKs packets that request it. For MX-MAC and X-MAC, the *acknowledgment request* field must be set to one. If no ACK is requested, the receiver simply turns off after the packet has been received..

7.5.4 Determination of the Switching Thresholds

In order to populate the MiX-MAC look-up table, we must compare the X-MAC, MX-MAC and SpeckMAC-D schedules and determine which is most appropriate to the current set of parameters on the network. We compare the three MAC schedules using three metrics: reliability of delivery, lifetime and throughput. The reliability comparison appears in Figure 7.13 and was discussed in the previous section.

7.5.5 Reliable Throughput or Goodput

In order to evaluate the throughput of the MAC protocols, we let one node send 100 packets to three different neighbors. $200\ ms$ after a transmission, packets are sent to the next neighbor, resulting in packet rates sometimes higher than $^1/_{t_i}$ for MX-MAC and X-MAC, as these protocols are interruptible and thus do not always transmit for the full t_i period. This should ensure that the application packet generating rate is not a limiting factor.

A second transmission mechanism illustrates the benefit of rescheduling transmissions immediately after a node has successfully sent a packet as explained in Section 7.3.2. We had a packet transmission occur $250\ ms$ after a successful communication, much like it would happen when a packet is deemed urgent. The $200\ ms$ and $250\ ms$ times are conservative settings, and their value could be easily lowered to mechanically increase the throughput.

Figure 7.14 shows the amount of correct bits transmitted in one second. For smaller values of t_i, X-MAC has the highest throughput. With larger values of t_i and for larger than $20\ B$ packets, MX-MAC performs best. This is because MX-MAC and X-MAC are interruptible, and thus take on average $^{t_i}/_2\ s$ to send a packet, and can attempt the next packet transmission $250\ ms$ later provided the source node is sending to different destinations. As the packet size increases, packets are received more reliably, which increases the throughput by up to 50%

Since X-MAC uses fixed sized advertisement packets ($40\ B$), its throughput increases linearly with the data packet size as shown by the figure. However, its performance is not equal to that of MX-MAC since MX-MAC's larger packets get transmitted more reliably (as shown in Figure 7.13). For a different reason, SpeckMAC based schedules are also linear: SpeckMAC can send only one packet per t_i period. When the packet size increases, so does the throughput.

These results teach us that MiX-MAC can select the MAC schedule that will yield the best goodput for a certain packet size and t_i value. The results in Figure 7.14 suggest that for $t_i = 1\ s$, the X-MAC schedule yields the best throughput for small packets (less than $40\ B$), while the MX-MAC schedule has the best performance for larger packets.

7.5.5.1 Lifetime for Unicast Packets

Figure 7.15 illustrates the differences in lifetime when increasing the time between CCAs. For values of m and n equal to $\{0.005; 0.005\}$, $\{0.005; 0.05\}$, $\{0.005; 0.5\}$, and $\{0.005; 1\}$ (for the curves from top to bottom), the maximum loss of lifetime is 6%, although on average, it is under 1%. However, this hides the fact that the receiver is more likely to correctly receive the packet under MiX-MAC than with SpeckMAC. This can be observed if we consider the case when a sender receives notification that a packet was correctly received through piggybacked ACKs. Figure 7.15(b) shows that in this case, there is no lifetime loss. Instead, the lifetime of the node peaks at a different value of t_i due to the change in energy expended to probe the medium.

Figure 7.16 shows the lifetimes for MX-MAC, X-MAC, and SpeckMAC schedules for various $\{n; m\}$ values and $15\ B$ unicast packets—n designates the rate of received packets, and m[2] the rate of transmitted packets. As per the simulations, SpeckMAC does not perform as well as MX-MAC and X-MAC. While this validates the simulations, Figure 7.16(a) shows that X-MAC performs best for smaller packets as opposed to previously shown. This holds only when the node is mostly sending. This is because the advertisement packets size went from $11\ B$ to $40\ B$, which increases the chance of being heard during a transmission, and thus prevents retransmissions. At the same time, an increase in packet size increases the energy consumption by only 3-4%. This is because the radio transmits for $t_i\ s$, whatever the packet size.

When the node starts receiving many packets as in Figure 7.16(b), this advantage is lost because a packet received under the X-MAC schedule costs roughly twice as much energy. The advantages of X-MAC are reduced further when the data packet size reaches that of the advertisement size because the advertisement packet is no longer easier to hear than the data packet.

This section shows that for packets smaller than $40\ B$, and for cases when the node is mostly sending, X-MAC allows the node to increase its lifetime. In other cases, MX-MAC leads to a longer lifetime. While the receiver does not get to pick the MAC schedule, cross-layer information, as provided by X-Lisa, allows the sender to select the MAC that will spare the energy of the node with the smallest remaining energy.

---

[2] m and n may be the same if the node is a relay that does not introduce new packets onto the network. $n = 0$ would typically designate a source node, and $m = 0$ a sink node.

The receiver does not need to be informed of any changes in MAC scheduling. Based on the packets received, the receiver knows which schedule the transmitter is following.

7.5.5.2 Lifetime for Broadcast Packets

Contrary to unicast packets, one MAC schedule consistently spares the energy of the node, over the range of packet sizes. Figure 7.18 shows that X-MAC performs very poorly, as shown by the simulations as well (Section 7.4.1.1). Figure 7.18 also shows that the lifetime increase provided by SpeckMAC is modest (2%) when the node is mostly sending, and larger (10%) when the node is mostly receiving.

These relatively small lifetime increases hide the fact that with the SpeckMAC schedule, the destination nodes are much more likely to correctly receive the packets (section 7.5.3.1). Thus, using the SpeckMAC schedule for broadcast allows for longer lifetime and more reliable communication.

The results for the broadcast case show that MiX-MAC should always select the SpeckMAC-D schedule: as it will enjoy small gains in lifetime, and it will greatly improve packet delivery reliability compared to MX-MAC and X-MAC.

7.5.6 MiX-MAC Achieves the Upper Bound of Node Lifetime

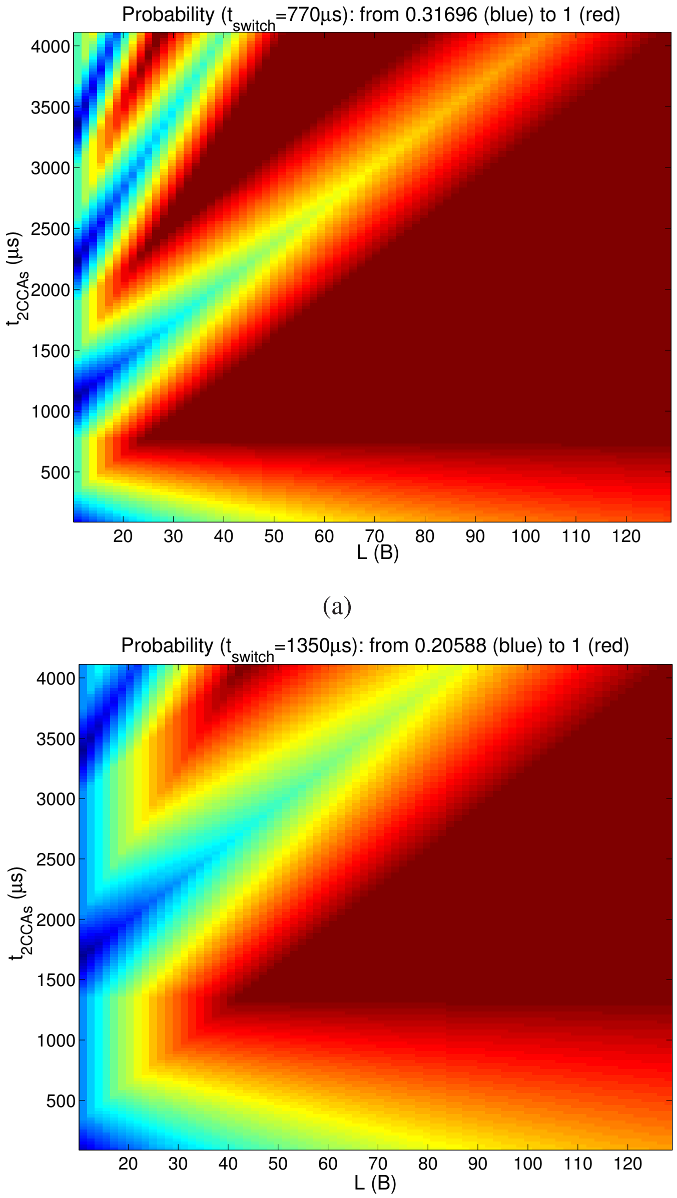
In order to show the benefits of MAC schedule adaptation, we present the lifetime of a node sending unicast packets of different size. MiX-MAC selects either the MX-MAC or X-MAC schedules based on the packet size.

7.5.6.1 Picking the right MAC schedule

The previous results show that all MAC protocols sacrifice performance in unicast mode to that of the broadcast mode or vice-versa. MiX-MAC performs well against every combination of parameters because it constantly picks the best MAC schedule for these parameters.

MiX-MAC adopts SpeckMAC-D's schedule for broadcast packets, and for unicast packets, it uses four axes to decide the appropriate schedule. These include t_i value, packet size, estimated ratio of transmitted vs. received packets, and the ACK requirements determined by the upper level protocols or services.

The simplified look-up Table 7.5 gives the optimal schedule as a function of several parameters. There exists an inherent trade-off between the size and complexity of the look-up table, and the granularity of switching MAC schedules over all the considered parameters.



(a)

(b)

Figure 7.12: Probability to successfully hear an ongoing stream of packets as a function of the packet size (L) and t_{2CCAs}, for an RX / TX switch time t_{switch} = (a) $770\ \mu s$ (SpeckMAC), and (b) $1,350\ \mu s$ (X-MAC / MX-MAC).

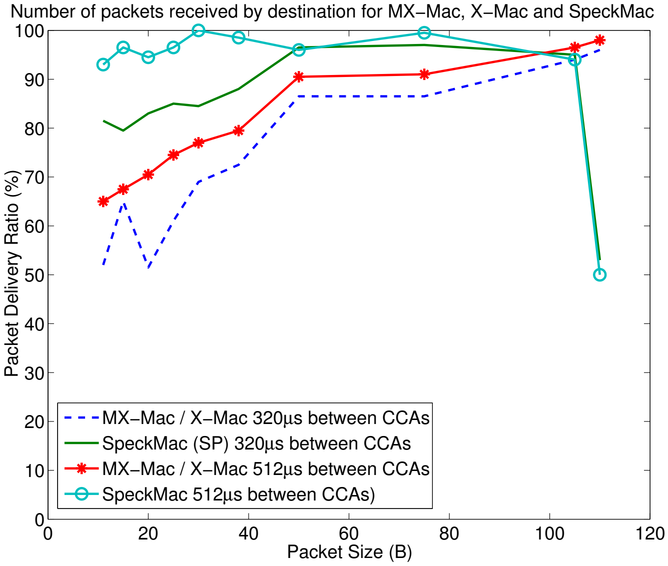
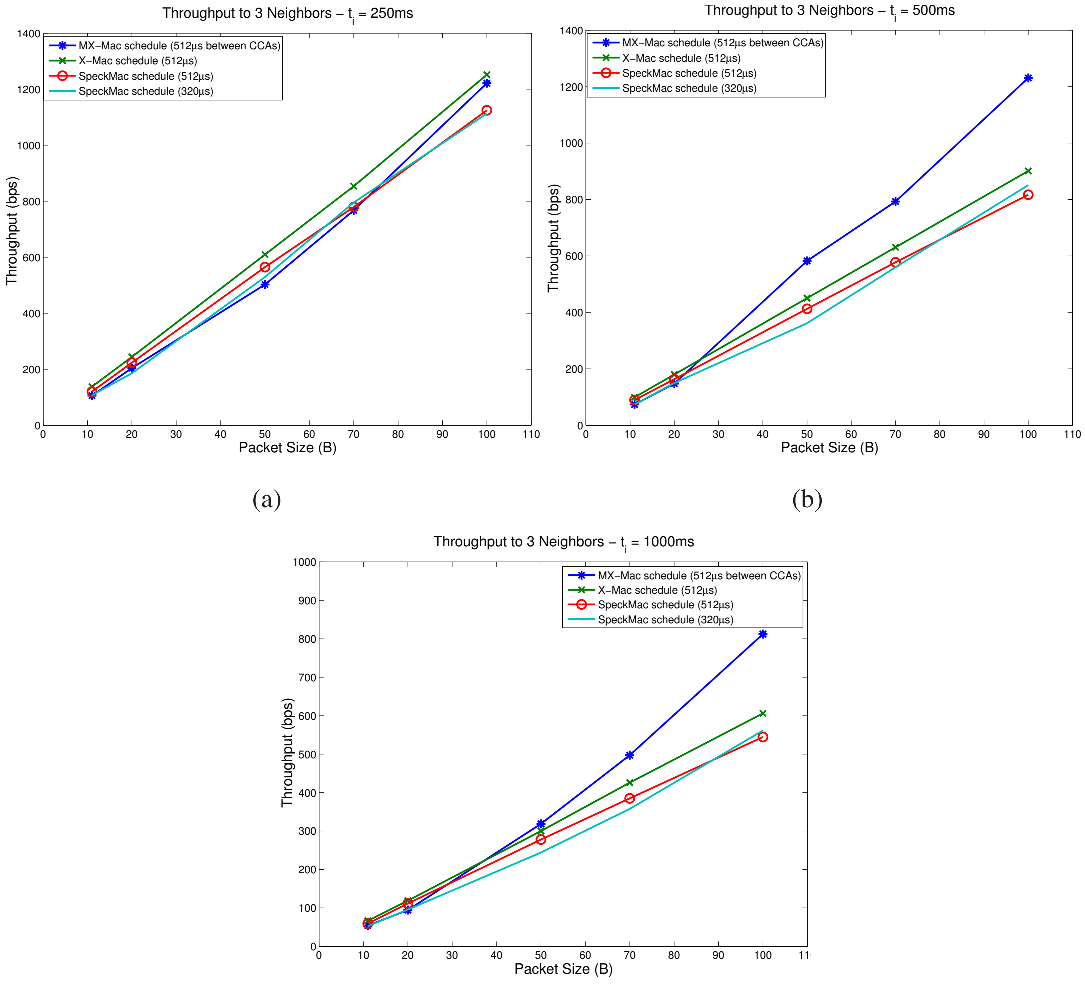


Figure 7.13: Comparison of the packet delivery ratio of the MAC schedules as a function of packet size and time between CCAs.

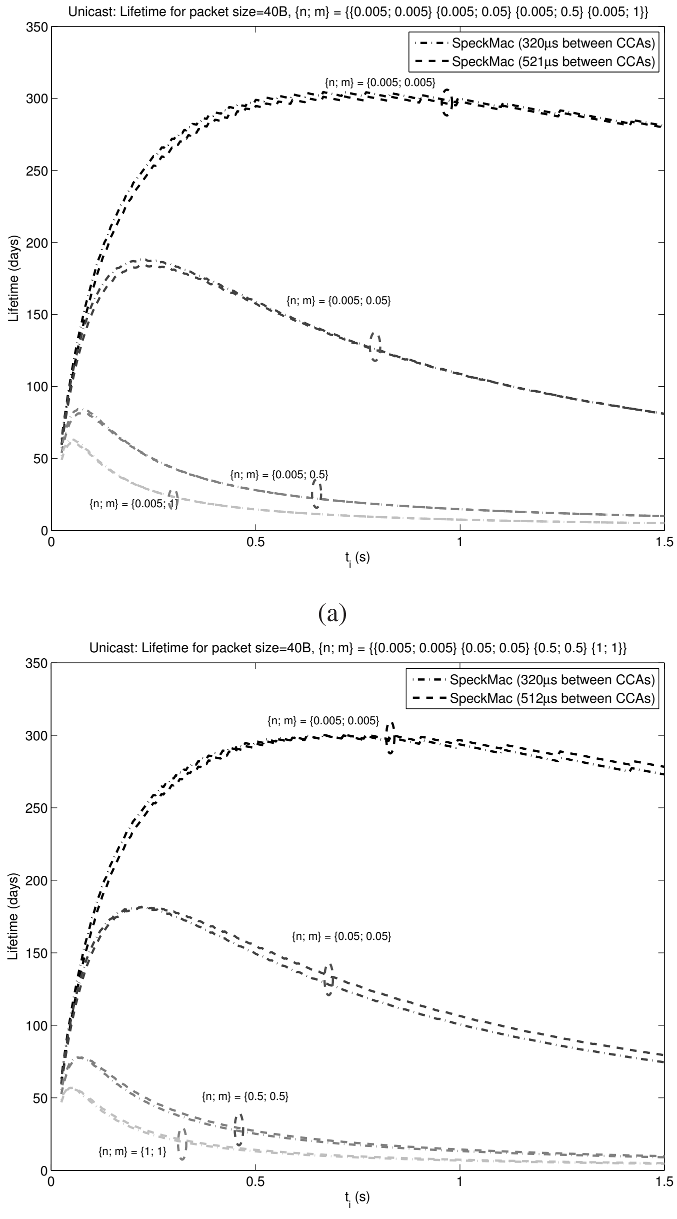


(a)

(b)

(c)

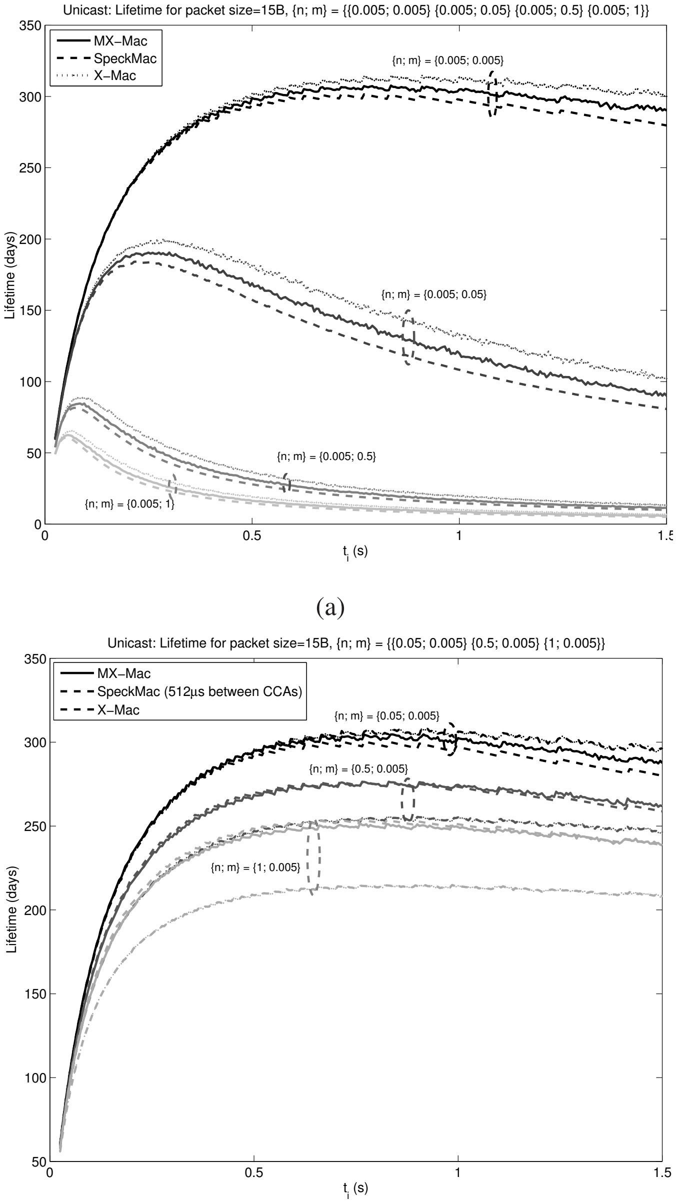
Figure 7.14: Throughput for t_i values of $250\ ms$, $500\ ms$, and $1\ s$.



(a)

(b)

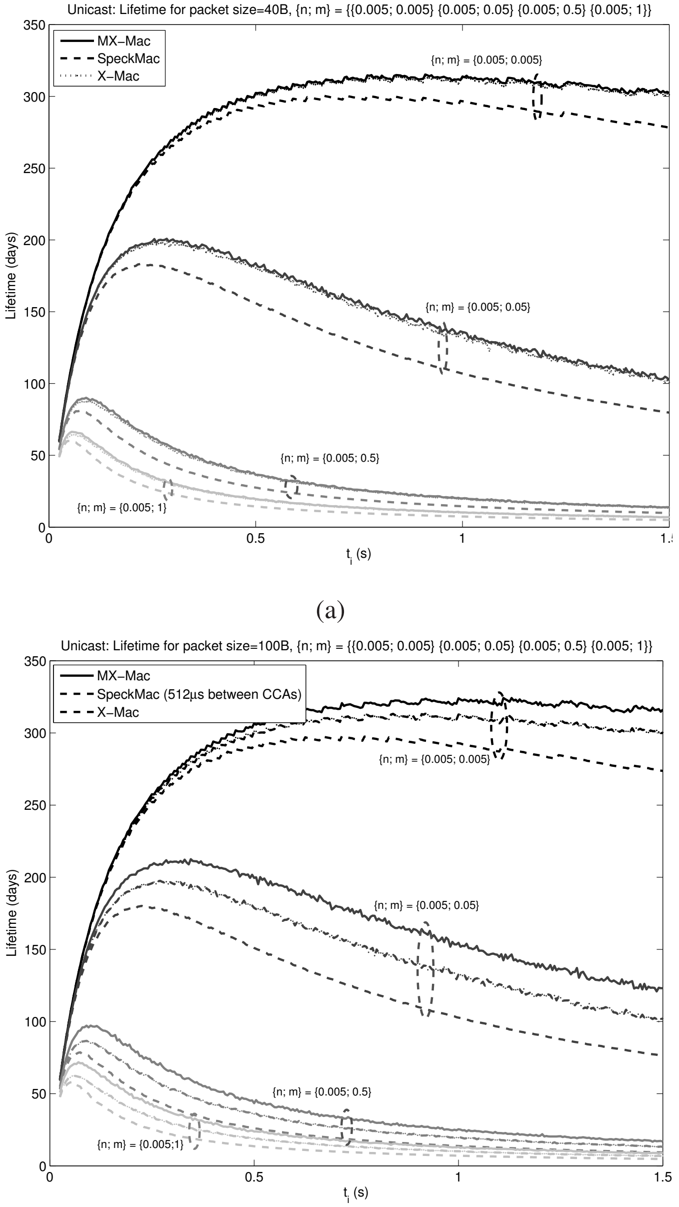
Figure 7.15: Comparison of the lifetime of SpeckMAC and MiX-MAC. The two schedules only differ by the time between their CCAs. In (b) we assume that the receiver notifies the sender of the reception of packets through piggy backed or stand-alone ACKs. (a) assumes no such mechanism.



(a)

(b)

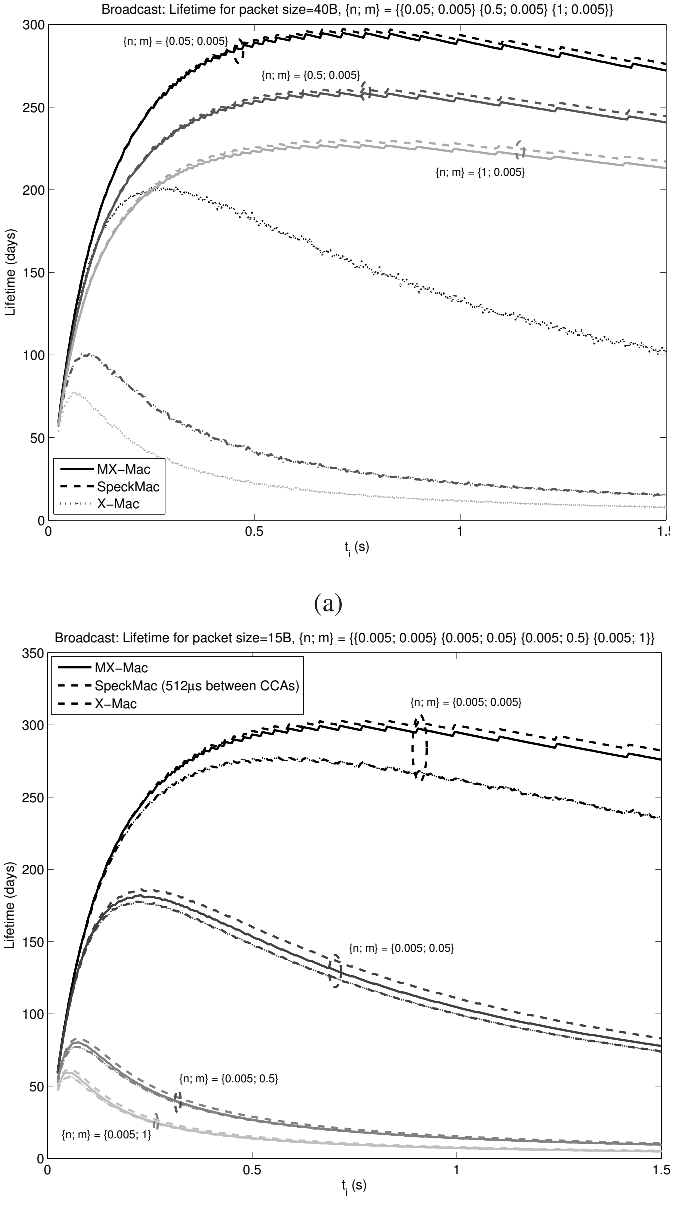
Figure 7.16: Comparison of the MX-MAC and X-MAC schedules for $15\ B$ unicast packets for scenarios where the node is mostly sending (a), and mostly receiving (b).



(a)

(b)

Figure 7.17: Comparison of the MX-MAC and X-MAC schedules for $40\ B$ unicast packets (a) and $100\ B$ packets (b) when the node is mostly sending.



(a)

(b)

Figure 7.18: Comparison of the MX-MAC and X-MAC schedules for $40\ B$ broadcast packets (a) when the node is mostly receiving, and $15\ B$ packets (b) when the node is mostly sending.

Table 7.5: Look up table to determine which protocol (SpeckMAC-D (S), X-MAC (X), or MX-MAC (M)) performs best in terms of lifetime.

Pkt size	Unicast				Broadcast	
	Mostly Receiving $m = 0.005$ $n =$		Sending and Receiving	Mostly Sending $n = 0.005$ $m =$		
	0.05		1	0.05	1	
15 B	X	M	X	X	X	S
40 B	X	M	X	X	X	S
80 B	M	M	M	M	M	S
120 B	M	M	M	M	M	S

7.5.6.2 Implementation Example

In this section, we present the node lifetime as a function of packet size in Figure 7.19 as an easy-to-read graph of schedule switching benefits.

The figure shows that MiX-MAC increases the lifetime by up to 30% compared to using the fixed schedules of X-MAC, MX-MAC, or SpeckMAC-D. While the mapping may not be perfect, the error of the point at which schedules are switched stems from the fact that schedules have very similar energy patterns at these packet sizes. The error in schedule switching is thus inherently small.

As can be seen in the figure, for the values of $\{m; n\}$ presented here, the lifetime may actually increase with the size of the packets sent. This is because a sending node is on for a fixed amount of time, and more reliable communications help avoid retransmissions as shown in Section 7.5.5.

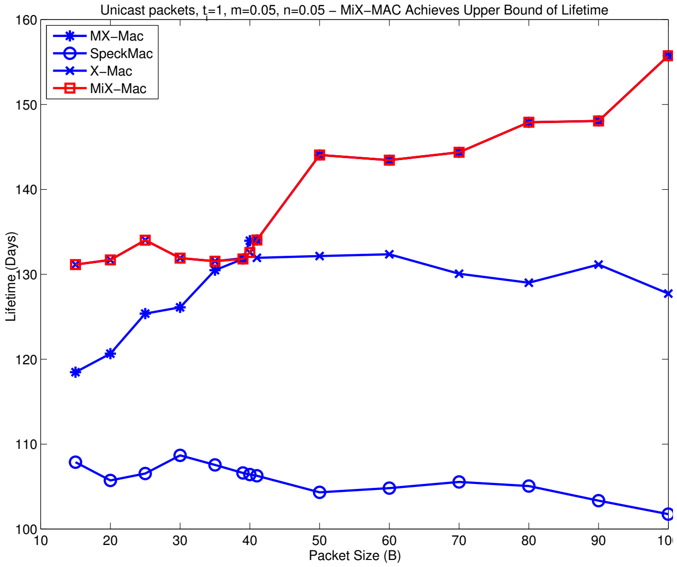


Figure 7.19: A simple mapping function lets the protocol switch between schedules in order to increase the lifetime.

7.5.6.3 Effects of erroneous estimates resulting in suboptimal scheduling decisions

Since MiX-MAC picks schedules from a pool of existing protocols, erroneous estimates for the optimal t_i (because of the wrong measurement of m or n for instance) would equally affect the performance of X-MAC, MX-MAC, and SpeckMAC-D.

However, if the lookup table were to point to an incorrect schedule, due for instance to an outdated or inaccurate estimate of the number of transmissions over receptions ratio, the network would simply operate at the level of performance of the chosen MAC protocol, without further degradation. For small estimate errors around the points where schedules are switched (the intersection in Figure 7.19), the difference between the energy consumption of the optimal schedule and that of other schedules is small. A small error around these points cuts lifetime by only a few percentages as our experiment suggests.

Elsewhere, a small estimate error has no effect: because the error is too small to impose a different schedule, the sending node picks the same schedule as the optimum. For very large estimate errors (for which the point of switching schedules is between the estimate and the actual value of a parameter), the resulting performance loss may be significant; however, large estimate errors (over 20%) should be rare by nature.

With this section, we saw that adapting the best-suited MAC schedule could increase node lifetime by up to 30%. The focus that we adopted was that of a node, without consideration for its neighbors. This meant that the transmit / receive schedules of nodes along a path were independent of one-another. We now shift focus to whole paths and try to synchronize nodes along routes in order to obtain further energy savings.

7.6 Discussion and Summary

We discuss the contributions and results of this work in this section.

- *MiX-MAC: Adapting the MAC Schedule to Conditions in the Network.* Existing MAC protocols employ identical schedules for both unicast and broadcast packet transmissions or, when impossible, simply modify their "unicast schedule" to work with broadcast packets. For instance, IEEE 802.11 cannot perform an RTS / CTS handshake for broadcast packets, and thus only utilizes CSMA for broadcast packets, regardless of the impact on lifetime or contention. We propose MiX-MAC, a MAC protocol that switches MAC schedules based on a set of easily obtainable parameters such as the unicast / broadcast nature of packets, their size, their numbers, etc. MiX-MAC chooses the MAC schedule best fit for the existing conditions in the network. This idea was made possible through the realization that many *LPL* MAC protocols were compatible and could be implemented with the *same* code, at the cost of no overhead.

- *Protocol Comparisons Against Several Parameters.* We identify three *LPL* MAC protocols that were compatible with one another and could be run with the same code: SpeckMAC-D, X-MAC, and MX-MAC. MX-MAC is a proposed modification of X-MAC to accommodate broadcast packets and limit false positive acknowledgement. In order to determine which MAC schedule MiX-MAC should adopt, we evaluated all three protocols through simulation and implementation in TinyOS against several metrics. Results show that no protocol performed consistently better in terms of lifetime. Parameters such as packet size, t_i value, and the ratio of packets sent vs. packets received have various impacts on a node's energy consumption. The ratio of n vs. m can be understood as burdens set on the sender or on the receiver. Since the stream of packet repeats of SpeckMAC-D cannot be interrupted, the brunt of the energy consumption is placed on the sender. SpeckMAC-D will thus perform better at lower t_i values for they allow shorter transmissions. Since X-MAC and MX-MAC can interrupt their packet streams, on average after ${}^{t_i}\!/_2$, they do not place as heavy a burden on the sender. Under the X-MAC schedule, a receiver needs to stay on approximately twice as long as with other schedules, thus resulting in a shorter lifetime when the node is receiving many packets.

If considered independently from packet delivery reliability, larger packets only marginally increase the energy consumed by a sender, although they have a greater impact on the receiver. The reason, specific to channel probing schemes, is apparent when considering the time for which the radio is on. Whether sending large or small packets, the radio needs to be active for t_i s and the time it takes to transmit a few (less than five) additional packets. The energy expended for sending larger packets stems solely from the packets sent *after* the t_i period, and is mitigated by the fact that the radio is in Tx mode more often for large packets.

The MAC protocol comparison results can be mapped to a look up table determining which schedule is best fit for conditions in the network and application requirements on delays and reliability. There is an inherent trade off between the correctness of MAC schedule selection and the table complexity.

- *Implementation Under TinyOS.* After running simulations using a popular analytical model, we implemented all MAC schedules in TinyOS. Doing so brought a wealth of important lessons. Among the most important teachings is the fact that TinyOS incurs significant delays for some operations. For instance, reading the RXFIFO turned out to be a very slow process relative to other instructions. Also noteworthy is the fact that sending automatic hardware ACK frames may expend *less* energy than abstaining from doing so. Automatic ACKs are sent promptly after a packet reception and require careful coding to avoid incorrect radio states. Also important is the time separating two (or more) CCAs, as this value may not fit all protocols or packet sizes. A programmer who knows that smaller packets will never be sent on the network may elect a slightly different time separation between CCAs.

- *Method to Accurately Measure the Energy Consumption.* We propose a new and accurate method to measure energy dissipation during complex operations such as typically found at the MAC layer. Our method allows us to measure the current draw from a mote without disrupting its operations. It also lets us precisely time each radio function. We were able to notice that every change in the TinyOS code could be felt in the current draw measurements.

- *Simulation Vs. Implementation Results Comparison.* We compare results obtained through simulation and through our TinyOS implementation. While the implementation results are closest to what end users would observe with current TinyOS deployments, the analytical simulations represent the results obtained for the radio without any operating system and may correspond to future implementations under an ultra-lightweight OS or faster CPUs.

  For broadcast packets, our simulations show a larger lifetime improvement than in the implementation. The difference originates from the time it takes TinyOS to revert to Tx mode after a packet transmission: this results in increased energy consumption. While in the analytical model, the time to switch to Tx mode has a ratio of 4-to-1 from X-MAC / MX-MAC to SpeckMAC-D ($736\ \mu s$ to $192\ \mu s$), this ratio is only 2-to-1 in TinyOS. The energy difference is further reduced by longer probes in TinyOS. However, investigation of packet delivery ratio shows that the SpeckMAC-D schedule is more reliable.

  For unicast packets, the main difference between simulation and implementation results stems from the size of the advertisement packets sent by X-MAC. Because 11B packets are typically hard to hear for nodes using two CCAs, we increased the size of advertisement packets to $40\ B$. This shifted ranges of t_i values and packet sizes for which X-MAC or MX-MAC allowed for longer lifetimes. However, in both sets of results, we observe that the optimal t_i value decreases with an increase in packet transmissions. We also noticed that in most cases SpeckMAC-D cannot outperform the lifetime obtained with the other two protocols, and that it suffers greatly from numerous packet transmissions for longer t_i values.

MAC schedule adaptation is not the only technique that can help dynamically adjust the behavior of *LPL* MAC protocols: in the following chapters, we investigate how the receive and transmit schedules of nodes running a subfamily of *LPL* MAC protocols can help lower their energy consumption and the packet delivery delays (Chapter 8). Later, we propose to utilize stochastic control theory to dynamically adapt the duty cycle and recast *LPL* MAC protocols for networks with high data loads (Chapter 9).

Chapter 8

Node Synchronization Along a Path

We propose to synchronize nodes along a slowly-changing routing path so as to minimize energy consumption and packet delay, *without* explicit scheduling between nodes or overhead of any sort. Only three *LPL* protocols can be selected to synchronize on unicast packets (and not lose this synchronization on broadcast packets): X-MAC, C-MAC and MX-MAC. These protocols form a subfamily of *LPL* protocols that can be interrupted by the receiver. For unicast packets, the sender stops its stream of advertisement (X-MAC / C-MAC) or data (MX-MAC) packets after receiving an acknowledgement frame. Sender and receiver can then be synchronized to wake-up sequentially within a short interval. Conversely, a non-interruptible MAC protocol such as SpeckMAC-D needs explicit notification within nodes to synchronize.

One of the major drawbacks of *LPL* MAC protocols is that they tend to place a significant burden on the sending node for medium to low duty cycles, even if we adapt the transmission schedule via MiX-MAC. The choice of a network-wide t_i value is a delicate decision: a programmer may wish to elect a large value, but such a low duty cycle may waste energy when transmitting packets, making it an unlikely choice. While researching MiX-MAC, we conjectured that certain *LPL* protocols could be synchronized in a way that would allow staggering wake-up schedules. This section details the various synchronization techniques that can be used to minimize delays and energy consumption. While synchronization along a path comes with virtually no overhead, the rare case when packets need to be exchanged on a bidirectional path requires a small amount of overhead (three extra packets per bidirectional path).

In the following, the term “interruptible LPL” or *int-LPL* refers to the subfamily of *LPL* MAC protocols whose stream of packets can be interrupted by an acknowledgement frame; to the best of our knowledge, these are limited to the three MAC protocols X-MAC, C-MAC, and MX-MAC.

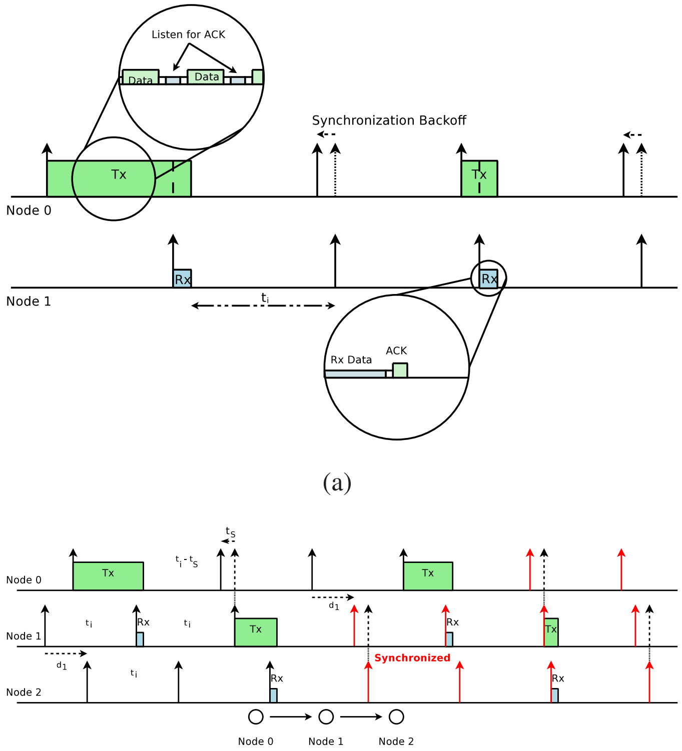
We chose to study MiX-MAC with only the MX-MAC schedule, although the results in this section can be easily extended to the whole family of *int-LPL* protocol schedules. Unlike X-MAC / C-MAC, MX-MAC is equally adapted to unicast and broadcast packets, and risks of false acknowledgement are smaller with MX-MAC. Most importantly, Section 7.5 showed that the small advertisement packets in X-MAC can be hard to hear, leading to rather poor link quality. Since our study applies to routing trees with at least two hops, the chance of packet delivery failure over one of the many hops on the routing path would be prohibitively high with X-MAC. We thus selected MX-MAC with $50\ B$ packets, although the principle of transmission / reception schedule adaptation holds for all *int-LPL* protocols. For this data packet size, the packet delivery ratio over one hop is close to 98$\%$, which translates into a packet failure rate of about 92$\%$ over a four-hop path, assuming independent links. For simplicity purposes, packets are delivered in a best-effort manner, and unsuccessful transmissions result in dropping the packet.

8.1 Synchronization Over a Unidirectional MX-MAC Link

8.1.1 Principle

Under the MX-MAC schedule, a node implicitly learns of the active schedule of its destination when it receives an ACK frame after successfully transmitting a data packet. It is this particularity that allows nodes running MX-MAC to synchronize.

Consider two nodes 0 and 1, with a unidirectional link from 0 to 1. After receiving a packet, node 1 sets its timer to wake up $t_i\ s$ later. The sending node 0 also sets a timer, in this case for t_i minus a small *synchronization back-off* $t_S > 0$. For the synchronization to take place, t_S must be greater than t_{Rx}, the time to receive a packet (one frame). This allows node 0 to wake up slightly before node 1 during the next rounds, thus reducing



(a)

(b)

Figure 8.1: Synchronization principle for (a) two (b) three nodes running an *int-LPL* protocol.

the time for which node 0 is transmitting. Figure 8.1(a) shows how the synchronization takes place for two nodes.

The requirement of unidirectionality is a minor one: WSNs are usually characterized by centrifugal broadcast packets (from the Data Sink to the peripheral nodes) and centripetal unicast packets (from the nodes to the Data Sink). Broadcast packets are commonly used to establish routes, refresh information about the end application, etc. On the other hand, unicast packets tend to flow from the periphery of the network to the data sink. For the nodes to correctly synchronize, the unicast packets must follow a slowly-changing route. Moreover, regardless of the direction taken by broadcast packets, the schedule for broadcasting packets under MX-MAC does not break the existing synchronization between nodes as the broadcast schedule may not be interrupted by an ACK frame.

The synchronization process for more than two nodes is less intuitive. Synchronization over multiple hops is achieved by following the same rules: a sender must always back-off by the same amount of time after it has successfully sent a packet. For the case of three nodes, full route synchronization is not achieved until after two packets have been sent, as illustrated in Figure 8.1(b).

8.1.2 Synchronization Process

Let $n = h$ be the number of hops from node 0 to node n. τ_k^j designates the time, modulo t_i, at which node k wakes up to probe the medium or send a packet, and after it has sent the j^{th} packet. At the beginning, wake-up times are separated by random periods of time. The effects of missing the beginning of a transmission (causing the receiver to receive the next frame in the stream) are negligible compared to t_S. When node 0 sends the first packet to node 1, both nodes synchronize and their wake-up times differ by the synchronization time t_S. The propagation of the first packet over the path leads to changes in the nodes' wake-up times as follows:

$$\tau_k^0 = \tau_{k-1}^0 + d_{k-1}$$
$$\tau_n^0 = \tau_0^0 + \sum_{p=0}^{n-1} d_p$$

When node 0 sends the first packet to node 1, both nodes synchronize and their wake-up times differ by the synchronization time t_S. $\tau_k^j = f(\tau_k^{j-1})$ designates the transformation that happens to the wake-up time of node k after it just sent the j^{th} packet:

$$\tau_1^0 = \tau_0^1 + t_S$$
$$\tau_n^0 = \tau_0^1 + \sum_{p=1}^{n-1} d_p + t_S$$

At the k^{th} hop along the path, the transformation f happens:

$$\text{From} \begin{cases} \tau_k^0 & = \tau_{k-1}^1 + t_S \\ \tau_{k+1}^0 & = \tau_k^0 + d_k, \end{cases}$$
$$\tau_k^1 = f(\tau_k^0) = \tau_{k-1}^1 + t_S + d_k - t_S = \tau_{k-1}^1 + d_k$$

At the last hop, nodes $n-1$ and n are separated by t_S:

$$\begin{aligned} \tau_n^0 = \tau_n = \tau_{n-1}^1 + t_S &= \tau_{n-2}^1 + d_{n-1} + t_S \\ &= \tau_0^1 + \sum_{p=1}^{n-1} d_p + t_S \end{aligned}$$

In other words, $\tau_n - \tau_0^1$ remains the same during the course of the first packet transmission from the source to the destination.

When the j^{th} packet is transmitted from node $k-1$ to node k we have:

$$\text{From} \begin{cases} \tau_k^{j-1} & = \tau_{k-1}^j + t_S \\ \tau_{k+1}^{j-1} & = \tau_k^{j-1} + d_{k+j-1} \end{cases}$$
$$\tau_k^j = f^j(\tau_k^0) = \tau_{k-1}^j + t_S + d_{k+j-1} - t_S = \tau_{k-1}^j + d_{k+j-1}$$

Thus, after the j^{th} packet, we have:

$$\tau_n = \tau_0 + \sum_{k=j}^{n-1} d_k + jt_S$$

The nodes are all synchronized when $\tau_0^n = \tau_n - nt_S$, that is after at most $j = n$ packets have been properly sent[1].

Once the path is synchronized, the delay can be expected to be equal to $t_S + (n-1)(t_i + t_S) + t_{Rx}$, as suggested by Figure 8.2.

---

[1] Synchronization will happen as long as the j^{th} packet reaches at least node $n - (j-1)$

This short analysis also shows that clock drift has little effect on path synchronization because this process uses only the nodes' relative—not absolute—positions in time. Synchronization is reinforced with every packet sent, making this protocol resilient as long as the clock drift is significantly smaller than t_S, which can be expected. The measured clock drift for the Tmote Sky is at most $5\ ppm$. This means that the relative drift between two motes is $\alpha_{drift} \leq 10^{-5}$. If $\mathcal{T}$ is the time between two packet streams, then we must have $\mathcal{T} < \frac{(t_S - t_{Rx})}{\alpha_{drift}}$. In our implementations, we commonly used a t_S value of $50\ ms$, leading to $\mathcal{T} < 3,600\ s$: in order to guarantee the proper preservation of the nodes' synchronized wake-up schedules, a unicast packet must be sent at least every hour on the path. For WiseMAC, $\mathcal{T}$ is only half that because nodes use absolute schedules to synchronize themselves. Therefore, nodes running WiseMAC should start sending $2t_S$ before their next-hop wakes up. The results in Section 8.4.1.5 take this into consideration.

8.1.3 Urgent Packets

Regular packets are forwarded in the next duty cycle after they have been received. On the other hand, urgent packets can be retransmitted immediately after they have been received. If a packet is marked as urgent (the implementation details are not relevant to, and beyond the scope of, this work) because of application or QoS requirements, the radio is kept on, waiting for the upper layers of the protocol stack to request sending the packet. When the "send" command is issued, the MAC protocol immediately starts the stream of packets. In order to be successful, the packet transmission must start before the next hop probes the channel. The delay associated with urgent packets is less than $t_i\ s$, thus greatly reducing the packet delivery latency over regular packets. Over synchronized paths, the delay of urgent packets is equal to $nt_S + t_{Rx}$.

The decision to send urgent packets within the same t_i period, but to exclude regular packets from immediate retransmission is a design choice motivated by practical implementation considerations. Support for urgent packets requires protocols from the Data Link layer to the Routing Protocol to collaborate and capably handle urgent deliveries. Today, this is rarely the case. Processing on each packet (snooping, queue reordering, next-hop calculation, loading the radio FIFO, etc.) must be very limited in order to meet the next-hop's wake-up time. If t_p is the processing time, we must have

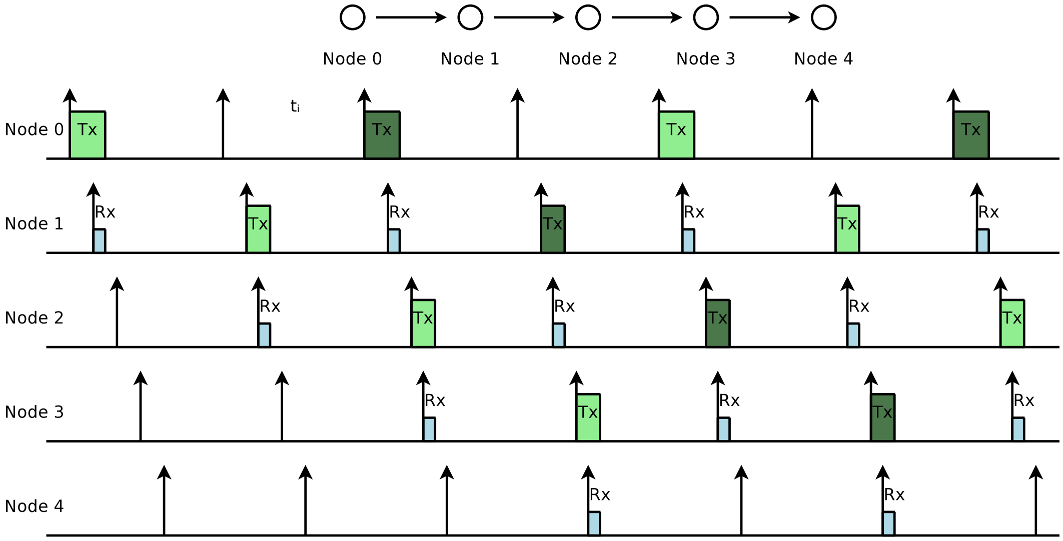


Figure 8.2: Node 0 pipelines packets and increases the packet rate.

$t_{Rx} + t_p < t_S$.

In spite of these caveats, a protocol designer may wish to treat all packets as urgent ones, and would thus benefit from very short delays.

8.1.4 Pipelining of Packets on a Synchronized Path

Because packet transmissions happen in a sequential way, packets can be pipelined over the path so that a packet is sent every $2t_i$, as illustrated by Figure 8.2.

Pipelining is only possible with synchronized nodes because if nodes are not synchronized they would interfere with one another and exacerbate the hidden node problem, common to all *LPL* protocols.

8.2 Synchronization Over a Bidirectional Path

Although uncommon in WSNs, some network topologies and applications may send unicast packets over paths that are in part or in whole bidirectional. This could be the case when several data sinks are deployed in the network. We developed an algorithm that coordinates bidirectionality on a path, although it induces overhead.

Let nodes i and j be the two ends of a sub-path $P_{i \to j}$. In order to synchronize nodes

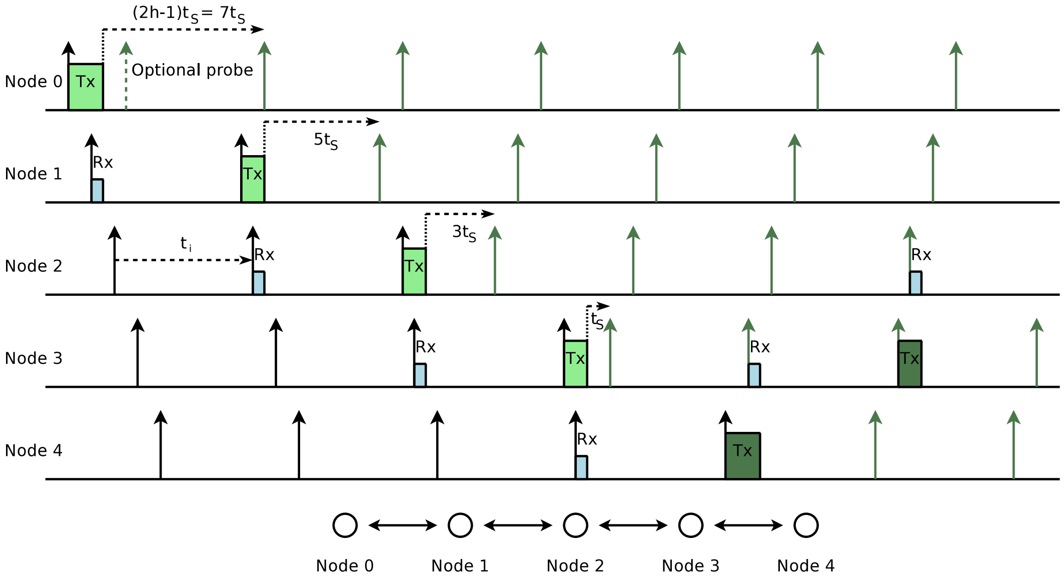


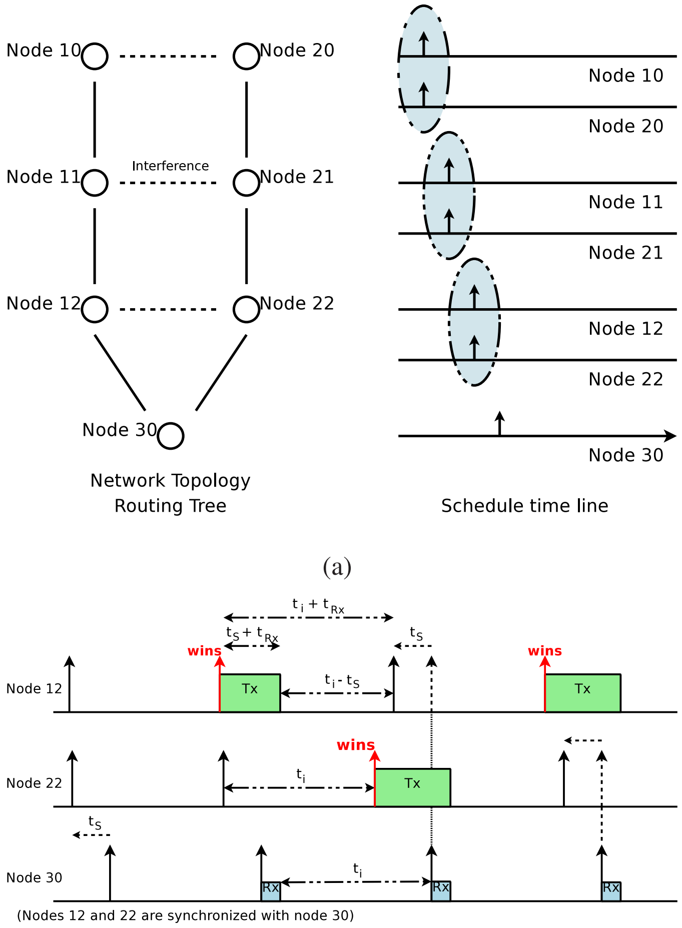
Figure 8.3: Bidirectional path synchronization time line.

over $P_{i \leftrightarrow j}$, the MAC protocol must allow packets to travel in only one direction at a time during synchronized rounds. Furthermore, cross-layer information such as the number of packets sent per round, and the number of hops from i and j is needed.

Each source node is allowed to send only a limited number of packets on the path at a time. Upon forwarding the last packet of the synchronized round, each node backs-off by $-t_S(2h_{k,\{i,j\}} - 1)\ s$, where $h_{k,\{i,j\}}$ is the number of hops from node k to i or j. The path $P_{i \to j}$ starts with unicast packets being forwarded from node i to j. If node j has to forward a packet to i, the link becomes bidirectional. A *Route Set-Up* header is added to the packet and contains the number of packets that node j plans to send during the round (only one at first), and the number of hops $h_{k,j}$ from node k to j. This process is inherently slow and breaks the synchronization of the nodes on $P_{i \to j}$. Node i replies with a *Route Reply* header containing the same information for i. Node i then transmits enough packets to restore synchronization between the nodes on $P_{i \to j}$ (*e.g.*, $h_{i,j} - 1$). The normal bidirectional operation between i and j begins, and the nodes exchange bursts of N_i and N_j packets, which depend on the packet rates of i and j and their queue size[2]. As it is forwarding the last packet in the burst, relay k back-offs by $-t_S(2h_{k,\{i,j\}} - 1)\ s$. Figure 8.3 illustrates this bidirectional synchronization process.

---

[2] N_i and N_j will be typically low in order to limit the packet delay.



(a)

(b)

Figure 8.4: (a) Synchronized nodes along two parallel paths: nodes $\{10, 11, 12, 30\}$ form one path, and $\{20, 21, 22, 30\}$ another one. The dotted lines indicate that the nodes can communicate with each other (and thus interfere). (b) Mitigation of the problem.

8.3 Path Synchronization With Multiple Sources

8.3.1 Synchronization Over Several Unidirectional Paths and Conflict Resolution

In some specific cases, the risk for packet collision still exists on synchronized paths. This is particularly true when a routing tree is formed of two or more (n_f) parallel branches: nodes i hops away from the destination tend to wake up at the same time, causing contention. This node configuration is illustrated by Figure 8.4(a).

The incidence of this problem depends on several factors such as the routing protocol (which may forward packets along parallel paths for robustness), the network topology (nodes from the same region may report highly redundant information if no packet fusion or aggregation strategy is employed) and the application (which may require high data rates from co-located sources).

Several techniques may be used to mitigate this phenomenon, including duty cycle reduction (Section 8.3.2 and Chapter 9), packet rate reduction, etc. However, the node schedule already offers a good solution to prevent collisions and to guarantee fairness among information flows under a light load.

If two neighboring nodes are part of two different synchronized paths as Nodes 12 and 22 are in Figure 8.4(a), they will attempt to send packets at about the same time. However, if node 12 can send its packet, it will wake up slightly after node 22. This is because node 12 will back off by t_S from the moment it receives an ACK frame, which happens after the time it takes to receive the data packet (t_{Rx})—a few tens of milliseconds. In effect, after successfully transmitting a packet, a node's wake-up schedule gets delayed by t_{Rx}, which separates it from other contenders and allows other flows access to the common destination. This process is not dependent on the number of flows converging to the same node, rather, t_{Rx} limits flow fairness as it should be above $\sim t_i \cdot n_f \cdot 2\alpha_{drift}$. Figure 8.4(b) illustrates this with a time line: after sending a packet, nodes 12 and 22 are separated by $t_{Rx}\ s$. They alternatively wake up before the other one as they send packets, guaranteeing fairness between the two branches of the routing tree.

However, when the data load increases beyond 1 packet every $t_i\ s$, it can no longer be accommodated, and multi-branch synchronization must be explicitly invoked.

8.3.2 Strategy

Let Figure 8.5 represent a three-hop network with two sources (marked by $*$) sending packets to a common destination $h = 3$. First, we define several terms used throughout in this section: a *branch node* refers to node k, as the location where several flows meet. Nodes from a packet source to the branch node form a *branch*. The nodes placed after the branch node are part of the *root* of the path.

The key idea to support the convergence of l flows to one node k consists in in-

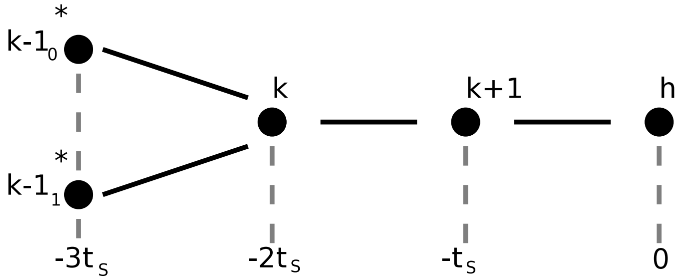


Figure 8.5: A multi-hop network with two sources $k-1_0$ and $k-1_1$.

creasing the duty cycle of root nodes ($\geq k$). The t_i value must be divided by l to accommodate fair access to node k by all sources $k-1_j$[3].

Upon receiving a new unicast packet, node k checks the ID of the previous-hop and adds it to a neighbor table if not present already. If a new source is detected, node k modifies the received packet to include a MAC header containing the value l, the ID $k+1$ of its next-hop, and the ID of the new previous-hop. The new t_i value is calculated as:

$$t_i^{new} = t_{i,k}^{new} = t_{i,k}\frac{l}{l-1}$$

The packet is then broadcast to all immediate neighbors. Nodes k and $k+1$ adopt the new duty cycle after forwarding the packet to their next-hop neighbor.

In the case of two flows converging toward node k, the node designated as the previous-hop in the broadcast packet header must delay its scheduled wake-up time by ${}^{t_i^{new}}/_2\ s$. This ensures that both nodes $k-1_j$ will not attempt to transmit a packet at the same time. For more than two flows, the broadcast packet need not specify who the new source node is: upon attempting to send a packet and finding a busy network, a node $k-1_j$ will back-off by ${}^{t_i^{new}}/_l\ s$ until all schedules are staggered.

---

[3] The notation $k-1_j$, where $j \in \mathbb{N}^+, j < l$, designates the previous hop of node k on branch j.

8.4 Simulation and Implementation of Synchronization Principles

8.4.1 Simulations

In this section, we explore the advantages and limits of node synchronization through Matlab simulations. We use the same accurate Matlab model for time and energy consumption as that of Section 7.5. Thus, the results provided by this section are those of an implementation reconstruction, rather than those of a simulation. However, to distinguish between direct results from our implementation, we use the words *reconstruction* or *simulation*.

In this section, ten nodes are randomly placed to form a multi-hop network. Unless otherwise specified, a source node sends packets at a rate of ${}^{1}/_{2}\ pkt.s^{-1}$.

8.4.1.1 Synchronization Principle

We simulated a scenario in which $t_i = 1.5\ s$, and the synchronization back-off t_S is $50\ ms$. Notice that the t_S value is about twice the reception time of a $50\ B$ packet. The duty cycle was chosen to be fairly low, since we expect that the improvements brought by path synchronization will allow the t_i value to increase.

Figure 8.6 shows that five nodes synchronize on the temporarily fixed path {1, 5, 4, 3, 10}. In the Figure, a triangular marker $\triangle$ represents a probe, $\circ$ a packet to send, $*$ a successful packet reception and $\times$ a failed one. After one packet, nodes 3 and 10 have staggered probes, nodes 4, 3 and 10 after the second packet.

This scenario also illustrates the behavior of the nodes when synchrony is lost: in the worst case scenario, it would take n packets to reconstruct a synchronized path. However, complete loss of synchronization is highly unusual because whenever a node fails to receive a packet from its neighbor, it simply wakes up t_i seconds later, *i.e.*, in the same relative time position.

8.4.1.2 Packet Delay

Next, we investigate the packet delay after the nodes have been correctly synchronized. We define packet delay as the time between the first attempt to send a packet and the

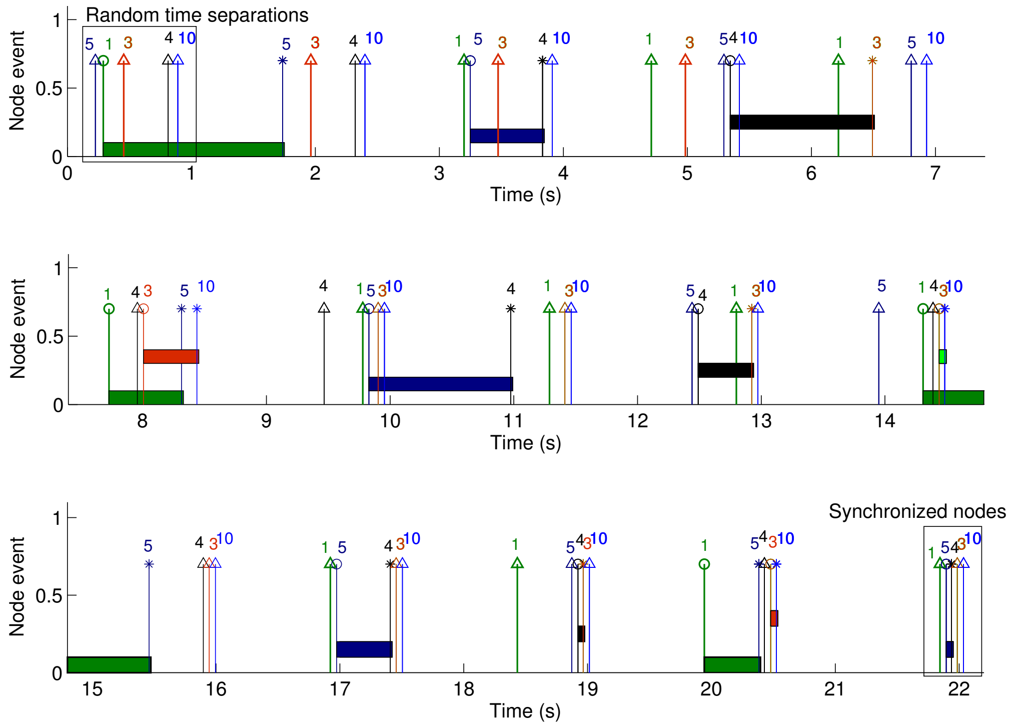


Figure 8.6: On the path $\{1, 5, 4, 3, 10\}$, the nodes synchronize correctly after only a few packets.

successful reception of this packet, noting however that the packet could have been created at most t_i s before the first transmission attempt. We consider that if a synchronized path is incapable of transmitting the required packet rate (the node queue keeps expanding), t_i needs to be lowered in order to accommodate higher traffic. We offer a solution to do so in Chapter 9.

8.4.1.2.1 Delay of Non-Urgent Packets Figure 8.7(a) shows the packet delay for the node configuration of the previous section. Because of the initial time differences between node schedules, the path was synchronized in only four packets. The packet delay then hovers around 4.74 s, which is approximately equal to $t_S + 3(t_i + t_S) + t_{Rx}$ (t_{Rx} is modeled by a random variable with a normal distribution).

The first packet is sent without any synchronization between the nodes and its delay is $5.8\ s$, a value that depends on the initial random wake-up times. Synchronization cut the packet delay by over 18%.

8.4.1.2.2 Packet Delay of Urgent Packets

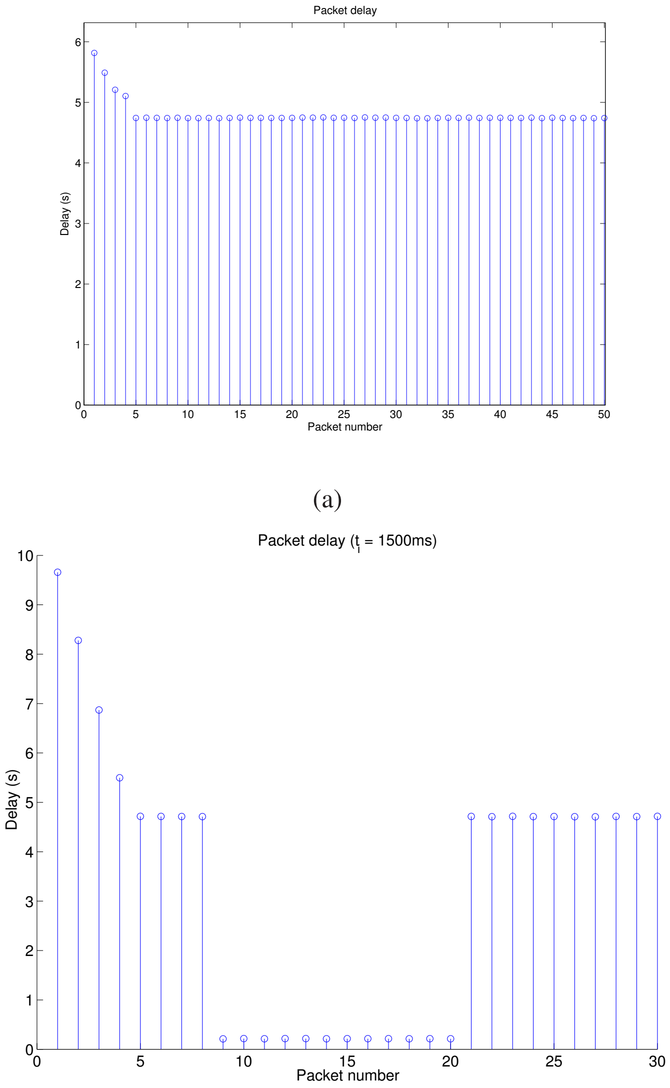
Figure 8.7(b) shows the packet delay of non-urgent and urgent packets. The very first packet is sent without any synchronization, and takes $9.7\ s$ to be delivered. Once synchronized, the delay is reduced by 50%. Packets 9 through 20 are marked as urgent: they are delivered almost immediately, with a delay of around $220\ ms$, which corresponds to $nt_S + t_{Rx}$ when $n = 4$. The simulation shows that the synchronization is not broken by urgent packets.

8.4.1.3 Bidirectionality

Although it is rarely expected in a WSN, bidirectionality may exist. We tested our algorithm over the same five-hop, now bidirectional, path $\{1, 5, 4, 3, 10\}$. We had to lower the t_i value to $0.5\ s$ in order to accommodate a larger load on the path. The larger issue of t_i control for *int-LPL* is addressed in Chapter 9, which offers a method inspired by control theory to dynamically set the duty cycle. In this section, the packet delay is defined as the time difference between the moment a packet is intended for delivery and the time when it is successfully received. This definition, slightly different from the other cases, allows results to reflect the time spent in the queue by a packet.

Figure 8.8 shows the delay for bidirectional packets sent by node 1 at a rate of $^1/_3\ pkt.s^{-1}$ and node 10 at $^1/_5\ pkt.s^{-1}$. The average packet delay comes at $3.6\ s$, mostly because packets must be queued while a node is not allowed to use the path. The very first unidirectional packet experiences a high delay ($4\ s$), and the following two must be queued while the bidirectional path is established. Note that the packet transmission from node 1 to 10 occupies the channel for the same amount of time as unidirectional packets (because the various nodes are synchronized). We can then consider that, on average, packets spend $1.8\ s$ in the queue for $1.8\ s$ of travel from node 1 to 10.

This shows that our algorithm is capable of maintaining synchronization over a bidirectional path, while keeping packet delay in check.



(a)

(b)

Figure 8.7: (a) Packet delay on the same path as Figure 8.6. (b) Same configuration, but packets 7 through 15 are marked as urgent.

8.4.1.4 Path Synchronization With Multiple Sources

We simulated the process of node synchronization along a path with multiple branches. Two sources 0_0 and 0_1 sent their packets to the same destination 4 at a rate of $0.5\,pkt.s^{-1}$ each. t_i was set to $1.5\ s$, enough to accommodate only one source. Retransmission mechanisms were put into place to replicate the likely design goals of a programmer: if a node already has more than two packets in its queue, it does not wait for the next

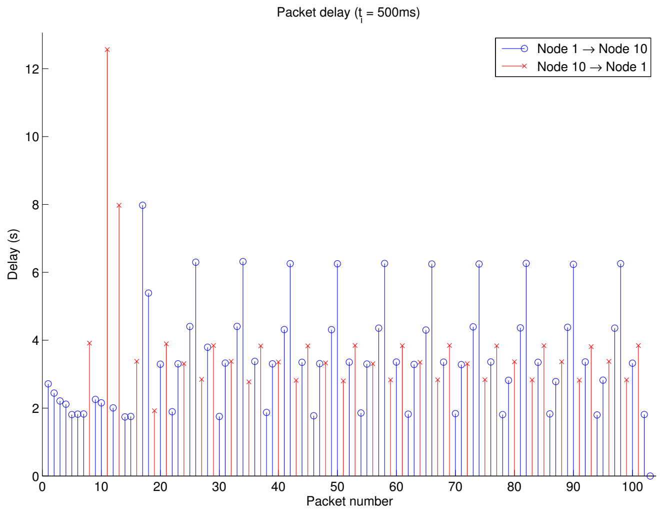


Figure 8.8: Delay on the same path as Figure 8.6 for bidirectional packets.

cycle to forward the packet in its queue.

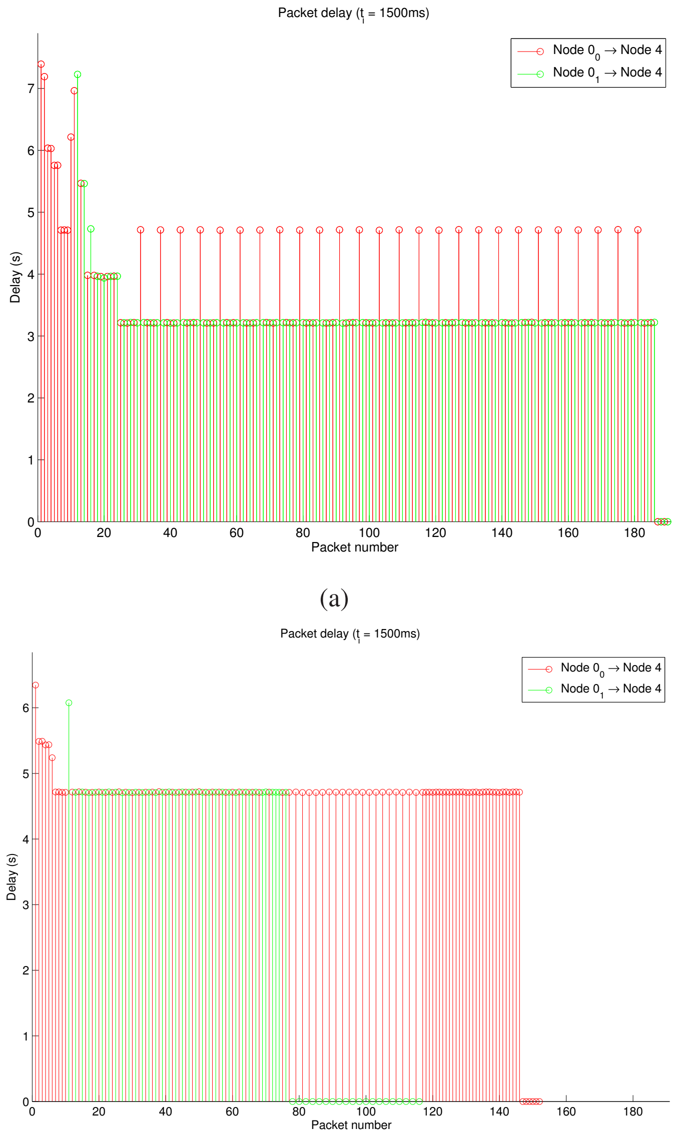
The simulation starts with only one source 0_0. After $20\ s$, the second source is turned on, forcing nodes on the root of the path to divide their t_i value. The branch synchronization process incurs delays until the destination 4 has doubled its duty cycle (it only takes a few cycles).

Figure 8.9(a) shows the packet delay when branch synchronization is supported. Figure 8.9(b) is the packet delay for the regular case.

In both cases, the initial synchronization process is slower because the source node sends packet before they are received at the destination 4 and have helped stagger every node's schedule. This can be avoided by considering all packets as urgent (they would be forwarded as soon as they are received).

Branch synchronization successfully delivers the combined data load of the sources with small packet delay. Most packets have a delay around $3\ s$, but some packets see their delay increased to $\approx 4.7\ s$. This is because some nodes on the path periodically empty their queue and do not forward the received packet within the same t_i. Urgent packets would all have the same delay.

In the case of regular synchronization, delivery of packets from either source is not necessarily denied, as was predicted by Section 8.3.1. Many packets reach the destination, whether from node 0_0 or node 0_1. From Figure 8.9(b), we can see that



(a)

(b)

Figure 8.9: (a) Packet delay for with (a) branch synchronization (b) regular synchronization.

nodes never completely empty their queue (the problem is particularly acute at node 1 whose queue kept increasing). Queuing causes many packets to be delayed, and over 200 s of simulation, 126 packets can be delivered, against 186 for the branch synchronization case. The increase in PDR is thus over 45%.

Table 8.1: Energy Consumption and Packet Delay.

		MX-MAC				WiseMAC	
		Sync		Non-sync			
		$\overline{Energy}$	$\overline{Delay}$	$\overline{Energy}$	$\overline{Delay}$	$\overline{Energy}$	$\overline{Delay}$
Parameter		(J)	(s)	(J)	(s)	(J)	(s)
h	1	1.59	0.065	12.73	0.76	2.53	0.13
	2	1.50	1.63	9.76	2.91	2.57	0.87
($t_i =$	3	1.49	3.21	9.37	6.10	2.60	1.52
$1.5\ s$)	4	1.54	4.91	9.00	10.63	2.61	2.28
	5	1.45	6.34	8.80	13.54	2.61	5.15
t_i	0.5	1.77	1.74	5.73	3.08	2.72	1.12
	1	1.67	3.26	8.99	6.62	2.62	2.11
($h = 4$)	1.5	1.54	4.91	9.00	10.63	2.61	2.28
	2	1.31	6.26	9.10	13.98	2.59	2.92

8.4.1.5 Energy Consumption

In order to fairly evaluate the energy benefit of node synchronization (and not just of *LPL* schemes), we compared the energy consumption of the proposed scheme with that of nodes running MX-MAC where neighbors wake up randomly within the t_i time interval.

Table 8.1 gives the average energy consumption of nodes on a path for MX-MAC and WiseMAC. Node 0 is furthest from node $n = h$, the destination. With our naming convention, h is also the maximum number of hops on the path. Because the destination node h is always receiving, its energy consumption is very low and depends only on the t_i value (as t_i increases, node h still uses the same amount of energy to receive packets, but it performs fewer channel probes). Thus, we excluded the energy consumption of node h from the average.

Typically, the non-synchronized nodes consume on average six times as much energy as the synchronized case. The reason for this difference is given by the average delay shown in the Table. When $h = 1$, the packet delay is about $^{t_i}/_{2} \approx 0.76\ s$ when the nodes are not synchronized, and $t_S + t_{Rx} \approx 65\ ms$ otherwise. This means that non-synchronized nodes must spend much more time with their radio active and transmitting than synchronized nodes. Consequently, the per-node average energy consumption greatly increases, by a factor of about ${}^{t_i}/_{2(t_S+t_{Rx})}$.

When the number of hops h is fixed and equal to 4, an increase in t_i reduces the per-node average energy consumption. This is only true when nodes are synchronized: if t_i increases, non-synchronized nodes must spend more time transmitting (for ${}^{t_i}/_2\ s$ on average). On the other hand, synchronized nodes must send for approximately $t_S + t_{Rx}\ s$, whatever the duty cycle. However, as the duty cycle decreases, the nodes have to spend less energy probing the medium, and thus synchronized nodes see their global energy consumption reduced. The same is true of WiseMAC, which reduces energy consumption for lower duty cycles.

Compared to WiseMAC, MX-MAC with synchronization consumes 30$\%$ less energy because it combines probes and transmissions, and because it uses hardware acknowledgements, which allow shorter packet reception times. However, the delivery delay of regular packets of MX-MAC is larger than for WiseMAC (it would be smaller for urgent packets).

8.4.2 Implementation on Tmote Sky

We implemented the principles behind node synchronization in TinyOS for the Tmote Sky platform. We present results from this implementation.

8.4.2.1 Methodology

Once the MX-MAC code was set onto the Tmotes, gathering results about packet delays appeared to be an intractable issue. In order to visualize synchronization, we needed to deploy a network of more than one hop. We chose to replicate the case of $h = 4$ (in a linear topology), often used in our simulations. In order to demonstrate synchronization, we opted to let Matlab—not the motes themselves—collect information about the packets. This is because in very time-sensitive MX-MAC, time stamping operations can be a delicate task for which CPU resources may not always be available on the motes. This however meant that all motes had to be in range of one-another and of the computer running Matlab, and had to be loaded with predefined neighbor graphs.

Yet, with this solution, motes that in a real deployment would not have to compete for the channel could now hear each other, artificially degrading the performance of our protocol. However, because of the nature of synchronized paths, packet collisions from motes placed at different levels of the routing tree did not compete for the medium at the same time, thus considerably alleviating this problem.

Our results are obtained from a mote receiving all packets transmitted over the channel and forwarding them to Matlab. Consequently, we cannot display channel probes since they are “silent” (the radio is in Rx mode only). We present results in spite of these caveats.

Finally, we cannot show the energy consumption of our implementation. This is because the Tmote sky can only measure its internal voltage through the ADC. This value is typically noisy, and the battery voltage does not evolve as a linear function of the energy remaining. Because the MX-MAC protocol is very energy efficient, the voltage drop over a measurable period of time is well within the natural ADC noise, with or without node synchronization.

8.4.2.2 Synchronization Principle

The goal of this section is to prove that node synchronization is practical and offers results in actual platform implementations.

Figure 8.10 shows that the motes on the 4 hop path $P_{0\rightarrow 1\rightarrow 2\rightarrow 3\rightarrow 4}$ successfully synchronize after the predicted number of packets. They also send packets in about $4.5\ s$ once they are synchronized, which is the delay predicted by the simulation model (within 4%, probably because Matlab starts time-stamping packets only *after* the first one has been received, and stops *before* the ACK frame is sent).

8.4.2.3 Urgent Packets

Next, we present the delay of urgent packets in Figure 8.11(a) and confirm the results obtained through the reconstruction model. Packets 7 through 10 are marked as urgent, and they see their delay hover between $766\ ms$ and $172\ ms$, the actual value of the delay being hard to measure because of the typically slow link between the mote and the PC. It also shows that urgent packets do not break the path synchronization.

The fast delivery of urgent packets is obtained through the immediate repetition of a received packet as shown in Figure 8.11(b). Small variations in the transmission times at every mote are natural, but the differences observed here are mostly due to the capture by Matlab, which does not receive ACK frames and can only deduce that a

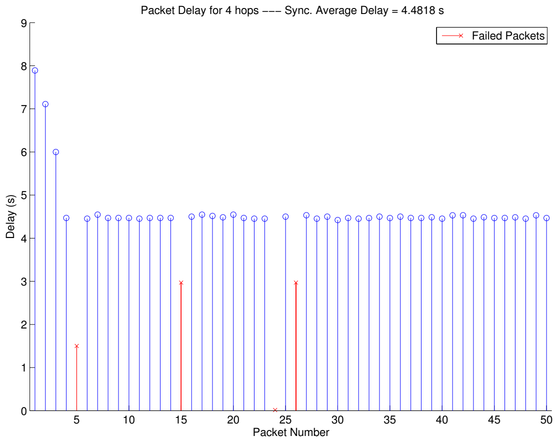


Figure 8.10: Packet delay over the same path of the implementation (failed packets do not reach the eventual destination and are not counted in the average delay).

transmission has ended after another has started. This causes some bars in the graph to be "glued" together.

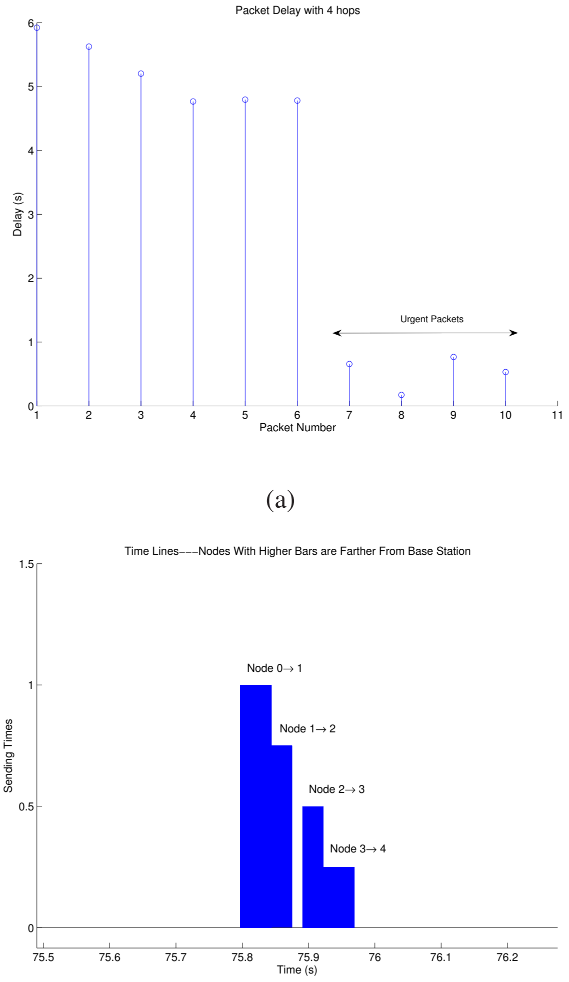
8.4.2.4 Packet Pipelining

Finally, Figure 8.12 shows the medium activity when the source node 0 sends a packet every $2t_i$. Thanks to node synchronization, packet transmissions can be staggered over non-continuous wireless links. The packet rate can then climb to ${}^{1}/_{2t_i}\ pkt.s^{-1}$, even though the packet delay remains the same.

This set of results shows that node synchronization over a temporarily fixed path is practical and works for the cases tested in simulation.

8.4.3 Combined Effects of MiX-MAC and Node Synchronization

Figure 8.13 shows the cumulative increases in node lifetime with MiX-MAC and synchronization. The conditions of the simulation are the same as in 7.5.6.2, although the results were averaged over fewer iterations. As in Figure 7.19, MiX-MAC achieves the highest lifetime for most packet sizes, with and without node synchronization, although synchronization greatly increases the lifetime of nodes running MiX-MAC (by up to 95%). When packets fail to be heard at the receiver, synchronization does not help in



(a)

(b)

Figure 8.11: (a) Delay of non-urgent and urgent packets over a synchronized path. (b) Time line of transmissions of an urgent packet.

any way: the packet will have to be retransmitted, regardless of when the receiver woke up—and missed— the packet. However, since MiX-MAC selects the protocol with the highest delivery reliability for each packet size, packet failures are less common, and synchronization can then play its full role of saving energy and reducing packet delay.

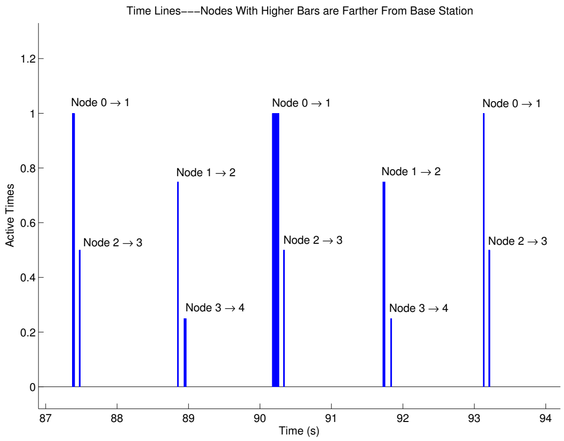


Figure 8.12: Packet pipelining: node 0 sends a new packet every $2t_i$. Transmissions end up being staggered.

8.4.4 Multi-Source Synchronization Implementation

We tested node synchronization for networks with multiple branches. Contrary to the results presented above, this technique requires overhead.

Our test network consisted of four nodes, with two sources (0_0 and 0_1) sending to node 1. The initial t_i value was $1.5\ s$. The TinyOS code was successfully tested for more sources, but since they are harder to read, these results are not shown here.

Figure 8.14 illustrates the process of path synchronization with two sources sending data packets to a common destination every $20\ s$ and $10\ s$. The Y-axis indicates the ID of the transmitting node. During the first half-minute, the path is established through a simple route discovery protocol and only one source is turned on. It takes two packets to synchronize the first source, which can be observed by the narrowing of the transmission bars. After $45\ s$, the second source turns on and sends its packet. It is immediately followed by a broadcast packet (sent over the full duration of the t_i interval). Immediately after, we can see that the schedules are staggered. After $75\ s$, node 0_0 sends a packet to node 1, immediately followed (${}^{t_i}/_{2}\ s$ later) by node 0_1. Node 1 then forwards both packets successively.

Figure 8.14(b) shows the packet delivery delay for both sources. Once the nodes are synchronized, the packet delay hovers around $1\ s$—except for each sources' fourth packet, because 0_1 sends its data before 1 can forward the packet from 0_0. The first

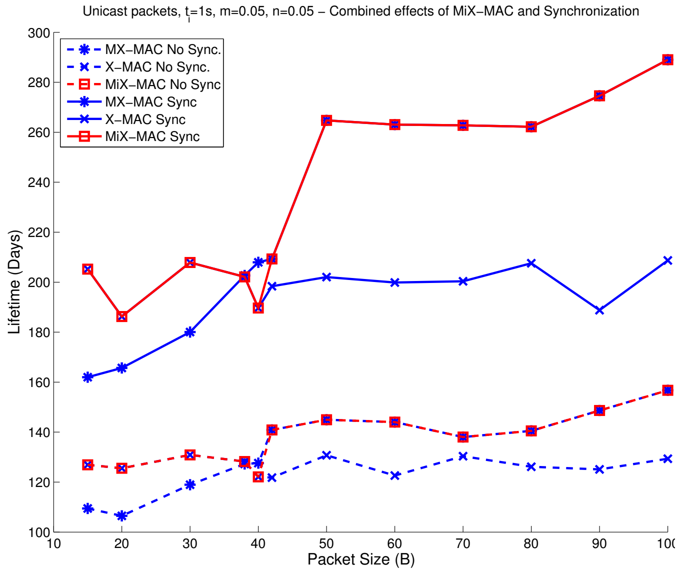
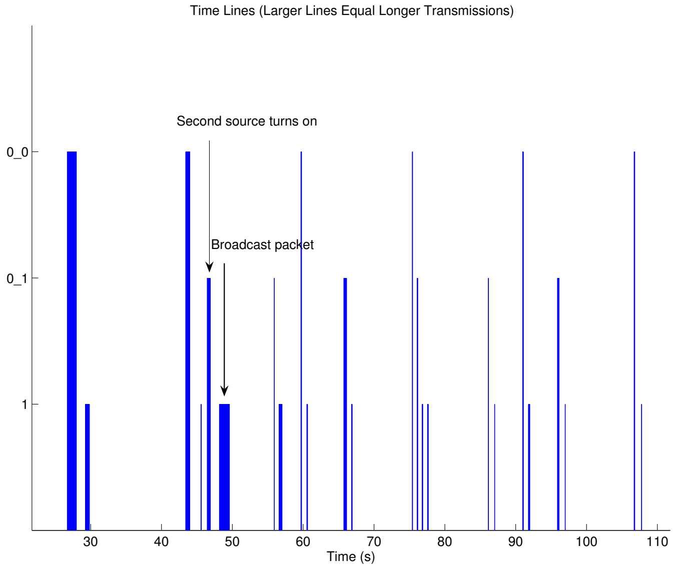


Figure 8.13: Comparison of the lifetime of nodes running MiX-MAC with and without node synchronization.

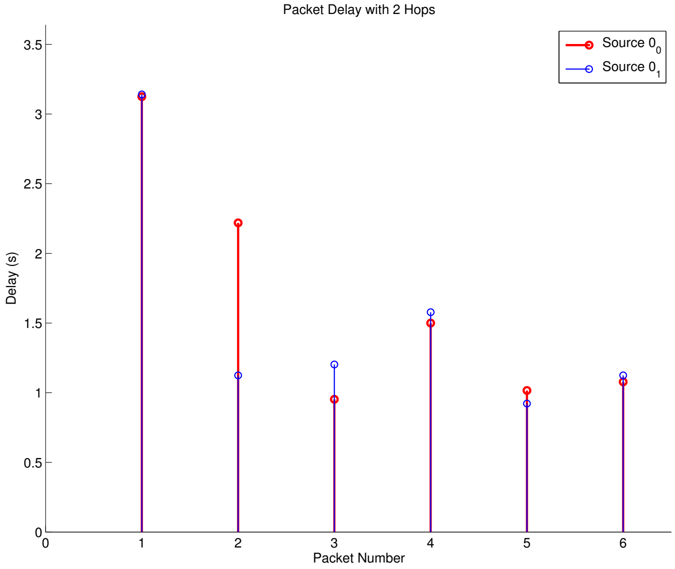
packets for both sources experience almost the same long delay (greater than $3\ s$), although for different reasons: when source 0_0 sends its first packet, nodes are not yet synchronized, and packet delivery is delayed by long transmission times. We see this delay being reduced in the following packet because node 1 synchronizes with its next-hop neighbor. The first packet of the second source, however, is delayed by the transmission of the broadcast packet indicating a new t_i value. Since synchronization is already in place on the existing path, source 0_1 is synchronized with node 1 with the first packet, which explains why the following packet (with ID 2) from node 0_1 experiences low delay.

8.5 Summary

We proposed and demonstrated a simple approach to synchronizing nodes on a temporarily-fixed path for the sub-family of *int-LPL* protocols. Through analysis, we proved that the path is automatically synchronized after $n = h$ packets have been sent from node 0 (the farthest) to node n. In other words, the requirement to have a fixed path is a weak one.



(a)



(b)

Figure 8.14: (a) Successful path synchronization for two nodes sending data packets to a common destination. The packet rates are $^1/_{20}$ and $^1/_{10}\ pkt.s^{-1}$. (b) The reduction in packet delay.

Synchronization of transmit / receive schedules has several benefits: it drastically reduces the packet delay, and it reduces the energy use at every node by a factor of about $^{t_i}/_{2t_S}$, removing the limit standing in the way of lower duty cycles.

In addition, we proposed several strategies to increase the packet rate and further reduce the packet delay. By pipelining packets over synchronized paths, we doubled the packet rate. Our approach also allows urgent packets to be delivered almost immediately, taking the delay from $t_S + (n-1)(t_i + t_S) + t_{Rx}$ to $nt_S + t_{Rx}$.

Although MiX-MAC and node synchronization may be implemented without the benefit of the other, their combined impact on node lifetime and packet delivery delay exceeds that of each protocol independently. This is because MiX-MAC may select the most reliable MAC schedule, which in turns greatly facilitates node synchronization.

Finally, we proposed a synchronization technique for tree networks with several branches, allowing each source to send a packet every t_i s.

While this chapter focuses on adapting the wake-up schedule of nodes along a path, we present a third way to adjust node behavior to conditions in the network by dynamically controlling the duty cycle of nodes running a *LPL* MAC protocol.

Chapter 9

Duty Cycle Control for Dynamic Adaptation of Low-Power-Listening MAC Protocols

In previous chapters, we sought to adapt the MAC schedule of *LPL* MAC protocols (Chapter 7) and the transmission / reception schedule of the nodes (Chapter 8). With these techniques, we can reduce energy consumption and packet delay. However, packet pipelining and branch synchronization techniques are not always enough to deliver all the packets in the network. In this chapter, we investigate how to control the duty cycle to accommodate a varying (and sometimes high) traffic load on the network.

We begin the study of t_i control by considering a one-hop link. As we saw in Chapter 7, longer t_i values favor receiving nodes, because longer t_i values lower a node's duty cycle while the time spent in *Receive* mode remains the same. On the other hand, nodes that are mostly sending can greatly reduce their energy consumption if the t_i value is low: they can stay in *Sending* mode for shorter periods of time. Consequently, there is a trade-off between the nodes at the two ends of a unidirectional wireless link. In addition, lower duty cycles often cause contention in areas of the network experiencing higher rates of packet transmissions. As Figure 7.1 shows, only one data packet can be transmitted per cycle, which can cause a node to miss the target rate m^* of packet transmissions. Alternatively, we could consider the delay it takes to transmit a packet in place of the number of packets successfully sent. Our model lends itself well to this substitution.

In [32], Jurdak et al. convincingly argue that a fixed t_i value does not fit WSN deployments where the node locations and traffic patterns are not uniform over the network. Because a fixed t_i value is decided *a-priori*, it would have to be set conservatively to accommodate areas in the network where traffic is expected to be heavy, thus forcing idle subregions to waste energy.

In this chapter, we borrow from control theory to propose a new approach to dynamically adjust the duty cycle of the nodes based on a small set of parameters. We begin with one-hop networks. The goal of our method is to minimize the energy consumed by the node with the lowest remaining energy (or the node which the application deems most important), referred to as node $\mathcal{N}$, while exchanging a target number of packets. If $\mathcal{N}$ is mostly sending, lowering t_i (increasing the duty cycle) will have no adverse effect on the target rate m^* of successfully sent packets, and it will reduce the energy dissipation for $\mathcal{N}$, so there is no need for t_i control. However, when $\mathcal{N}$ is mostly receiving, lowering the duty cycle (increasing t_i), while reducing the energy dissipation for $\mathcal{N}$, will cause packets to be dropped. This is the conflict that we propose to arbitrate. Our method can also be extended to control the energy consumed by both the sending and receiving nodes on a wireless link. More generally, we provide a methodological framework that can be applied to control other aspects of the network as well.

We generalize this method to multi-hop networks, starting with only one data source, as is often the case when source selection is performed. We propose a path synchronization scheme that, among other many benefits, reestablishes linearity in the system. We then lift the last restriction to have only one source through the study of t_i control for multi-hop networks with several sources. To successfully control the duty-cycle, a new path synchronization technique is introduced.

9.1 Estimation and Control for Multi-variable Systems

Because low duty cycle schemes tend to create contention and delays, a node wishing to send m^* packets may not be able to do so in a timely manner. Let us consider a one-hop network with various flows among neighbors. Node A wants to send m^* packets to node B in a certain time period $\mathcal{T}$, where node B is designated as node $\mathcal{N}$, a critical

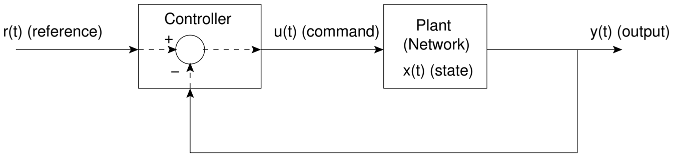


Figure 9.1: Representation of the system with input / output and its controller.

node for the application, or one with very low remaining energy. Unfortunately, the medium is sometimes occupied by other transmissions. If node A only gets to send $m < m^*$ packets, it may elect to increase its duty cycle. When the duty cycle is larger than its optimal value, node $\mathcal{N}$ wastes precious energy, and may wish to scale back its duty cycle (t_i increased). The control of the duty cycle to send m^* packets is the subject of the first part of this section. We use $t_i(t)$ to designate the time-varying nature of t_i.

9.1.1 Background

We provide mathematical background common to all our work throughout this chapter. Although its adaptation to different networks changes, the theoretical foundation is the same from one-hop to more complex networks.

9.1.1.1 Generalities

We start by assuming that the system we wish to represent and control is mostly linear. For instance, the relationship between energy consumption and t_i is linear, as energy consumption grows linearly with the number of probes done per second. Likewise, the number of packets received is mostly linearly related to energy consumption.

Figure 9.1 illustrates the system at hand. The network is represented by a "plant" that reacts to an input $u(t)$ by producing an output $y(t)$, which it tries to match to a reference $r(t)$. A controller modifies $u(t)$ so as to obtain the desired output $y^*(t) = r(t)$. In order to do so, the process under control can be defined by its state $x(t)$. A deterministic noisy linear process can be represented in its discrete form as follows [79]:

$$x(t+1) = Ax(t) + Bu(t) + Cw(t) + w(t+1) \tag{9.1}$$

where $x(t+1)$ designates the value of the system state at time $(k+1)\mathcal{T}$ and w is the noise. $\mathcal{T}$ represents the period between re-evaluations of the control $u(t)$.

For controlling $t_i(t)$, we can set $y(t)$ to $m(t)$ (the number of packets that are successfully sent at time t) and $u(t)$ to the $t_i(t)$ value at time t. The objective value $y^*(t{+}1)$ becomes $m^*(t+1)$, the desired number of packets to be transmitted at time $t+1$.

Because the fundamental characteristics of the system (A, B and C) and its state $x(t)$ cannot be *a-priori* known, the system's output must be estimated using an internal parameter θ and a history of p values of $\{x(t)\}$ (or $\{y(t)\}$) and $\{u(t)\}$ values stored in ϕ.

9.1.2 The Adaptive Regulator

In this first part of our work, we would like to control t_i to send the target number of packets m^*. We introduce a SISO (single variable) estimator and controller.

9.1.2.1 Stochastic SISO Estimator and Controller

We begin with the formulation of our goal, *i.e.* the minimization of the expected error between the desired output at time $t+1$, $y^*(t+1)$ and the actual output at time $t+1$, $y(t+1)$, which is mathematically represented by the following:

$$J = E[(y(t+1) - y^*(t+1))^2] \tag{9.2}$$

This control problem is referred to as linear-quadratic: the system dynamics are linear (Equation 9.1), but the cost function to be minimized (Equation 9.2) is quadratic. Because the system response contains a random component (the exact wake-up timing between two neighbors), we study a system estimator and controller for the stochastic case.

First, and as suggested in [79], we introduce the following notation for time delay:

$$x(t-1) = q^{-1}x(t)$$

We can write the system as:

$$y(t) = ay(t-1) + bu(t-1) + cw(t-1) + w(t) \tag{9.3}$$

$$\Leftrightarrow (1 - aq^{-1})y(t) = bq^{-1}u(t) + (1 + cq^{-1})w(t) \tag{9.4}$$

From [79], Equation 9.3 can be put in the form:

$$C(q^{-1})y^0(t+1|t) = \alpha(q^{-1})y(t) + \beta(q^{-1})u(t) \tag{9.5}$$

where

$$\begin{cases} C(q^{-1}) = 1 - aq^{-1} + q^{-1}g_0 = 1 + (g_0 - a)q^{-1} \\ \alpha(q^{-1}) = g_0 \\ \beta(q^{-1}) = b \end{cases}$$

and y^0 represents the next value taken by y and g_0 is a constant.

The control law is thus shown to be:

$$u(t) = \frac{y^*(t+1) + (g_0 - a)y^*(t) - g_0 y(t)}{b \neq 0}$$

Let $g_0 - a = c$,

$$u(t) = \frac{y^*(t+1) + cy^*(t) - (a+c)y(t)}{b} \tag{9.6}$$

which is the minimum variance control, also the control law used in [33]. It follows easily that Equation 9.6 minimizes the mean-square error function J.

Next, we define the ϕ and θ vectors as:

$$\phi(t)^T\theta(t) = \hat{y}(t+1)$$

where $\hat{y}(t+1)$ is the estimated system output at time $t+1$. As a starting point, we chose to keep only the previous values of the input and output, or $p = 1$. From Equation 9.6, we use the two vectors:

$$\phi(t) = \begin{bmatrix} y(t) \\ u(t) \\ y^*(t) \end{bmatrix} \quad \theta(t) = \begin{bmatrix} a+c \\ b \\ -c \end{bmatrix}$$

The estimator can be computed using the Normalized Least-Mean-Square Algorithm (NLMS) [79] [80]:

$$\theta(t+1) = \theta(t) + \frac{\mu(t)\phi(t)}{\phi(t)^T\phi(t) + \omega}[y(t+1) - \phi(t)^T\theta(t)] \tag{9.7}$$

where $\mu(t)$ is a scalar, and ω should be chosen to avoid a division by zero when $\phi(t)^T\phi(t)$ is null. With our notations, $\phi(t)$ is thus the values of the output $y(t) = m(t)$, the command $u(t) = t_i(t)$ and the target $y^*(t) = m^*(t)$. The tuple $\{a, b, c\}$ is estimated using Equation 9.7.

New t_i values are computed periodically. During each round (of duration $\mathcal{T}$), the number of packets successfully transmitted since the last t_i update (i.e., $m(t)$) is recorded.

9.1.2.2 Application to Our Estimator

The system control can be approached by estimating the system first, and using the system model to find the input value that minimizes the predicted output.

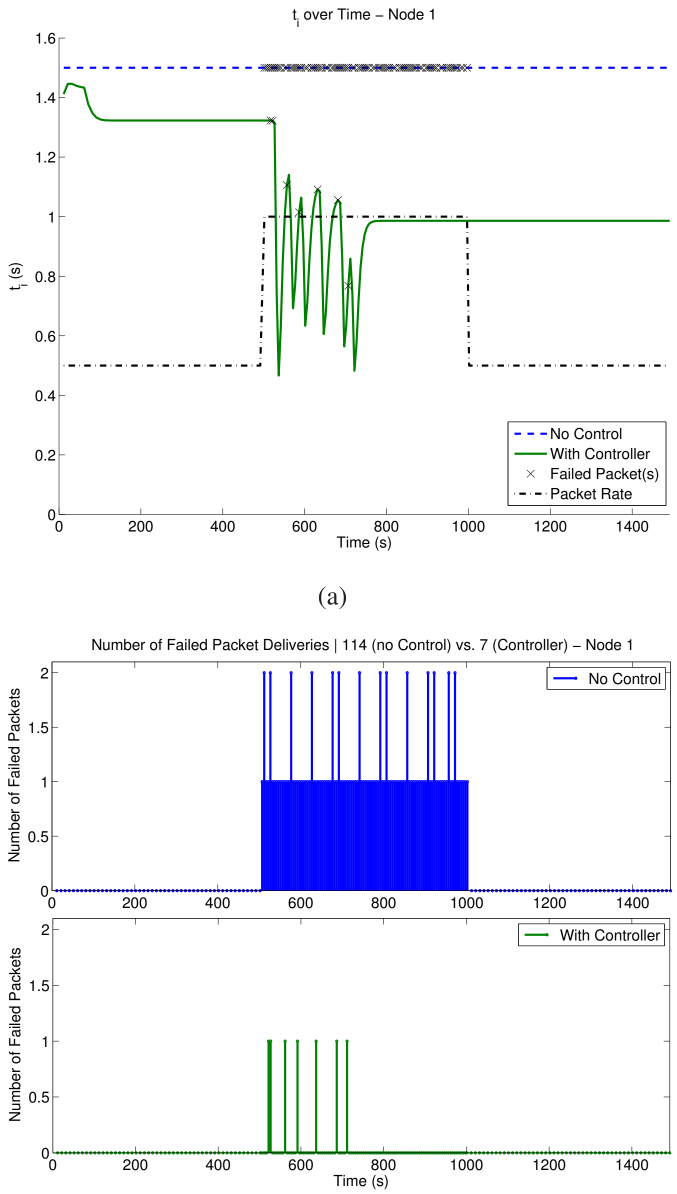
Preliminary results show that, while the estimator is able to correctly predict the system output, the control law tends to decrease the value of $t_i(t)$ when $m < m^*$. This behavior is in fact to be expected as the system should decrease its duty cycle to increase the number of packet transmissions. Unfortunately, since J carries no consideration for energy use, t_i never increases, even after the number of packets to be sent has reached the target ($m = m^*$). The reason is that the error between $\hat{m}$ and m^* is zero, which does not modify the value of the controlled input $u(t) = t_i(t)$. Figure 9.2(a) illustrates this problem. At $t = 500\ s$, the packet rate increases to one packet per second, causing t_i to decrease due to packet losses. However, a few seconds later, the packet rate decreases to its original value of 0.5 packet per second, yet t_i does not increase again.

Figure 9.2(b) shows the packet loss in this scenario, where the number of dropped packets is reduced by over 94% by the duty cycle controller. However, since the duty cycle is much higher than required after the adaptation when the traffic rate is reduced, the energy consumed by the controlled scheme is larger than the uncontrolled one. Consideration must be given to the energy consumed ϵ, which is an incentive to lower the duty cycle. We note $\epsilon^*(t+1)$ as a target energy consumption at $t+1$.

In this now multi-variable case, we decided to estimate both the number of packets sent and the consumed energy separately. For m and ϵ, the ϕ and θ vectors are:

$$\begin{aligned}
\phi_k^m &= \begin{bmatrix} m_k & \dots & m_{k-p} & t_{ik} & \dots & t_{ik-p} & m_k^* \dots m_{k-p}^* \end{bmatrix}^T \\
\theta_k^m &= \begin{bmatrix} a_0^m & \dots & a_{p-1}^m & b_0^m & \dots & b_{p-1}^m & c_0^m \dots c_{p-1}^m \end{bmatrix}^T \\
\phi_k^\epsilon &= \begin{bmatrix} \epsilon_k & \dots & \epsilon_{k-p} & t_{ik} & \dots & t_{ik-p} & m_k^* & \dots & m_{k-p}^* \end{bmatrix}^T \\
\theta_k^\epsilon &= \begin{bmatrix} a_0^\epsilon & \dots & a_{p-1}^\epsilon & b_0^\epsilon & \dots & b_{p-1}^\epsilon & c_0^\epsilon & \dots & c_{p-1}^\epsilon \end{bmatrix}^T
\end{aligned}$$

where $a, b, c \in \mathbb{R}$ are the estimator coefficients. We chose $p \gtrapprox 3$, a value that allows the estimate for ϵ and m to be accurate, while being still manageable in limited memory space.



(a)

(b)

Figure 9.2: (a) Evolution of $t_i(t)$ as the packet rate increases and then decreases when only packet loss is considered. (b) Packet loss in the same scenario.

9.1.2.3 Cost Minimization

As per Section 9.1.2.2, the controller should minimize a cost function with a packet loss and an energy component. We tried to combine the two costs in various ways, including taking the maximum, the sum, and the weighted sum of the costs. The latter offered the swiftest and most stable response from the network. Thus, the controller attempts to minimize the following cost function J:

$$J = (m^{*+} - \hat{m}_{k+1})^2 + K_\epsilon(\epsilon^{*+} - \hat{\epsilon}_{k+1})^2 \tag{9.8}$$

where $\begin{cases} \hat{m}_{k+1} = \phi_k^{mT}\theta_k^m \\ \hat{\epsilon}_{k+1} = \phi_k^{\epsilon T}\theta_k^\epsilon \end{cases}$, m^{*+} and ϵ^{*+} designate the target values of m and ϵ at time $(k+1)\mathcal{T}$. K_ϵ is a weight given to the energy component of the cost function in order to indicate a preference to save energy (large K_ϵ) or to strictly meet the number of packets to be sent (small K_ϵ); for instance, K_ϵ can be chosen in $[2; 20]$. The control law finds the value of t_i that minimizes J.

Taking the derivative of J at time $k\mathcal{T}$ (we omit the k index notation for clarity), we obtain Equation 9.6 for our application:

$$t_i = \frac{\theta_p^m(m^{*+} - \sum_{i\neq p}^{u}\phi_i^m\theta_i^m) + K_\epsilon\theta_p^\epsilon(\epsilon^{*+} - \sum_{i\neq p}^{v}\phi_i^\epsilon\theta_i^\epsilon)}{(\theta_p^m)^2 + K_\epsilon(\theta_p^\epsilon)^2}$$

where the i-index value on ϕ_i and θ_i are the i^{th} value of these vectors, and u and v are the number of elements in ϕ_k^m and ϕ_k^ϵ ($u = 2p$ and $v = 3p$).

In order to smooth the response of the system, we adopt a conservative update policy $\bar{u}$ for the duty cycle with the following set of rules:

$$\begin{cases} \bar{t}_{ik+1} = \bar{t}_{ik} + \alpha(t_i - \bar{t}_{ik}) \\ \bar{u}_{k+1} = f_\delta^\Delta[\bar{t}_{ik+1}] \end{cases} \tag{9.9}$$

where $\bar{t_i}$ is the smoothed t_i and

$$f_\delta^\Delta[x] = \begin{cases} \delta & \text{if } x < \delta \\ \Delta & \text{if } x > \Delta \\ x & \text{otherwise} \end{cases}$$

δ and Δ are the minimum and maximum values that t_i can ever take, and can be set to $0.1\ s$ and $5\ s$ as reasonable values.

$\alpha \in \mathbb{R}$ is the slope of the update of t_i and helps stabilize the system response, which would otherwise be unstable because of steep variations of the reference $r(t)$ (the desired number of packets for instance) and delays in the feedback. A large α (*i.e.,* close to 1) aggressively updates t_i and incurs oscillations before reaching a determined value. On the other hand, if α is close to 0, no oscillations can be discerned but t_i is slow to reach its eventual value. Poor choices of α may cause energy waste or packet loss. The command used to control the duty cycle is in fact $\bar{u}$ as a smoothed output is critical to a physical network.

9.1.3 Evaluating the Target Energy

9.1.3.1 The Evaluation of ϵ and ϵ^*

In some cases, the system designer may want to minimize the consumed energy and choose $\epsilon^* = 0$. The risk incurred by this approach is that the duty cycle will tend to be lowered, even below a reasonable value—one that strikes a balance between the number of lost packets and energy consumption. This could be desirable when designing a system that needs to respond faster to lower energy consumption, and that can tolerate repeated packet losses.

In other systems, an acceptable energy consumption value has to be evaluated so that $t_i(t)$ does not consistently increase past a reasonable value. This target energy has critical importance as the system will have a tendency to stabilize around the value of t_i that yields this energy consumption, provided all packets are correctly sent. The control problem thus becomes a linear quadratic tracking ("LQ tracking") problem where the output of the network must match the energy (and packet delivery) reference.

We chose to evaluate the target energy as the sum of several basic operations (channel probe, packet reception, etc.) for which we precisely measured the energy consumption via a data acquisition board on the Tmote Sky platform. We evaluate the target energy as the minimal energy that can be expended during a round of $\mathcal{T}$ seconds:

$$\epsilon^* = max(0, m^* E[E_{Rx}] + E_{PD}(t_{\mathcal{T}} - m^* E[t_{Rx}])) \qquad (9.10)$$

where $E[E_{Rx}]$ is the expected energy spent to receive a packet, E_{PD} is the energy consumed by the radio for one second of power down mode, and $t_{\mathcal{T}}$ is the duration of a feedback round $\mathcal{T}$. The target energy ϵ^* assumes that each packet is sent every t_i s, and that no energy is wasted on probing a clear channel. It contains no information about other transmissions in the neighborhood as packet loss is taken into account in the first element of J.

9.1.3.2 An Alternative Solution to Evaluating the Consumed Energy

Because it may be impractical to evaluate the energy consumption components ϵ and ϵ^*, an alternative solution consists in increasing the command periodically if all the packets have been successfully sent and received. Once the maximum t_i value to correctly send m^* packets is reached, the t_i may no longer be increased. This reduces the complexity of the system to only one component m.

The relative simplicity of this method is offset by the slower nature of the response to increase t_i when the data load diminishes. In our work, we prefer evaluating the energy, but we show in our later results that such a method gives satisfactory results.

9.1.4 Algorithm for t_i Control

We implemented the previous theoretical foundations in Matlab; Algorithm 4 presents the pseudo-code of the controller used to command the network. The initialization of the algorithm variables includes assigning a starting value to the ϕ and θ vectors. ϕ can take the initial values of m^*, t_i and ϵ^*, while θ is initialized with values between -1 and 1. For instance, an increase in t_i translates into a decrease in m and ϵ of node $\mathcal{N}$, and thus the corresponding weights in θ are negative.

In our implementation, we chose an initial $\alpha = 0.01$ and then adjust α to be 0.2 after three iterations of the controller to prevent large oscillations during the first rounds of the estimators. Our network consists of 10 nodes, all in range of one another (the medium can be occupied by only one node at a time). We evaluate the new command $\bar{u}$ every $\mathcal{T} = \frac{5}{\text{packetRate}}$ seconds: for instance, if a node sends packets at a rate of 2 packets per second, the controller will run every 2.5 seconds. The feedback period $\mathcal{T}$ can be increased to reduce overhead, although a large value could cause the network adaptation to be sluggish—or worse, instable.

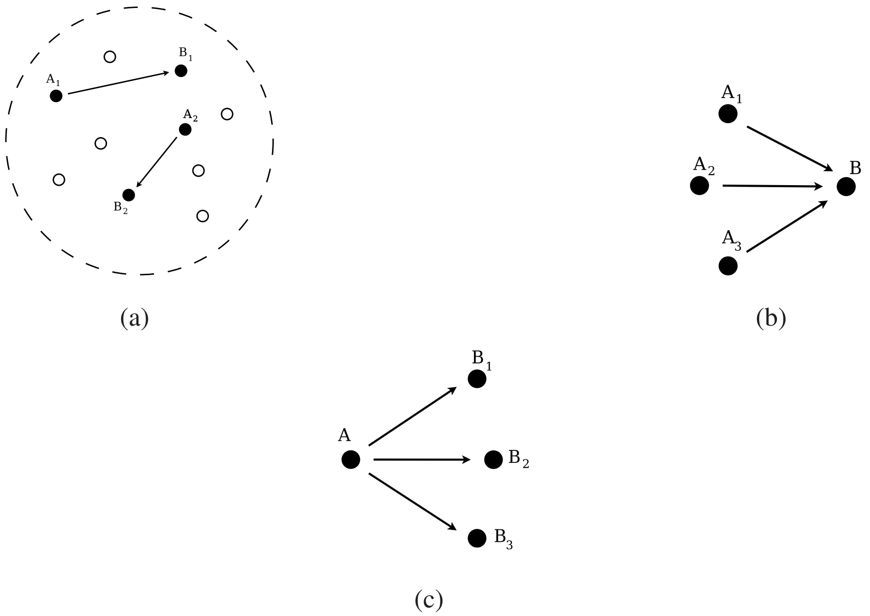
9.1.5 Preliminary Guidelines for an Implementation

For an implementation on real platforms, nodes need to periodically exchange information about their remaining energy and the new t_i values. Nodes should inform their neighbors (using broadcast packets for instance) of their remaining energy in order to determine the link node $\mathcal{N}$ whose energy should be spared. After $t_i(k+1)$ has been

```
Variable initialization:
$\phi^m = \begin{bmatrix} m^* & 0 & 0 & t_i & 0 & 0 & m^* & 0 & 0 \end{bmatrix}^T$
$\phi^\epsilon = \begin{bmatrix} \epsilon^* & 0 & 0 & t_i & 0 & 0 & m^* & 0 & 0 \end{bmatrix}^T$
$\theta^m = \begin{bmatrix} 0.95 & 0.1 & 0.1 & -0.5 & -0.1 & -0.1 & 0.3 & 0.1 & 0.1 \end{bmatrix}$
$\theta^\epsilon = \begin{bmatrix} 0.95 \ 0.1 \ 0.1 \ -0.5 \ -0.1 \ -0.1 \ 0.3 \ 0.1 \ 0.1 \end{bmatrix}$
for ever do
    $m^* = f(\text{packetRate}, \mathcal{T})$
    $\epsilon^* = f(\text{radio}, m^*)$
    $\theta^m \mathrel{+}= \frac{\mu^m}{r^m}\phi^m(m - \phi^{mT}\theta^m)$
    $\theta^\epsilon \mathrel{+}= \frac{\mu^\epsilon}{r^\epsilon}\phi^\epsilon(\epsilon - \phi^{\epsilon T}\theta^\epsilon)$
    $u = \frac{\theta_p^m(m^{*+} - \sum_{i \neq p}^{u} \phi_i^m \theta_i^m) + \theta_p^\epsilon(\epsilon^{*+} - \sum_{i \neq p}^{v} \phi_i^\epsilon \theta_i^\epsilon)}{(\theta_p^m)^2 + (\theta_p^\epsilon)^2}$
    $\bar{u} = f_{0.1}^{5}[\bar{u} + \alpha(u - \bar{u})]$
    $\phi^m = \overrightarrow{\phi^m} + \begin{bmatrix} m & \bar{u} & m^* \end{bmatrix}$
    $\phi^\epsilon = \overrightarrow{\phi^\epsilon} + \begin{bmatrix} \epsilon & \bar{u} & m^* \end{bmatrix}$
    Where $\vec{\cdot}$ is a matrix shift operator
    $r^m \mathrel{+}= \phi^{mT}\phi^m$
    $r^\epsilon \mathrel{+}= \phi^{\epsilon T}\phi^\epsilon$
end for
```

algorithm 4: Control pseudo-code for $p = 3$.

calculated, node A (the sending node) should communicate this new t_i value to node $\mathcal{N}$ using $t_i(k)$ and by piggy-backing the new value onto packets. For the family of *LPL* MAC protocols, bigger packets incur no overhead as the radio remains in *sending* mode for the same period of time (t_i s) regardless of the packet length. Both nodes can then start using the new calculated duty cycle. Node A estimates the energy consumed by node $\mathcal{N}$ to receive its packet *without* requiring $\mathcal{N}$ to report it. The reasons to proceed in this way are threefold: we were able to closely measure and model the energy required to receive packets—it has a Gaussian distribution with an average that is a function of packet size. Moreover, node $\mathcal{N}$ would *also* have to evaluate the energy consumed because $\mathcal{N}$ may be running other processes (packet processing, sensing activity, packet aggregation) that draw energy but are not relevant to the link A—$\mathcal{N}$. A third reason, although very hardware dependent, is that platforms like the Tmote Sky can only mea-



(a) (b) (c)

Figure 9.3: (a) Nodes are in range of one another. (b) A receiver has several descendants. (c) A sender has several destinations (rare case).

sure their battery voltage, which can be mapped to remaining energy but does not yield a sufficient precision when energy consumption is small. Errors made to evaluate the energy consumption are modeled in the noise component of the system.

We can also quantize the values taken by t_i in order to limit the duty cycle updates on individual links: node A would modify the t_i value only if the change is greater than a quantization step.

In WSNs, the directions of packets is usually fixed over a small period of time (*e.g.,* $\sim 10\mathcal{T}$) and centripetal: packets tend to travel from peripheral nodes toward the base station. Thus, most packets travel from node A to node B, with occasional (generally broadcast) packets going in the opposite direction. If nodes A and B are possible choices for node $\mathcal{N}$, node A can elect to minimize the consumed energy at both nodes. Our method works equally well by estimating the energy consumption at both A and B, although the t_i value tends to be noisier.

If a node is the receiving end of multiple links, as illustrated by Figure 9.3(b), it should adopt the smallest t_i value t_{iL} calculated by its descendants in order to receive all packets successfully. The amounts of energy wasted on the links using a lower duty cycle are negligible because the sending nodes will stop their packet transmissions after half t_{iL} s on average—protocols like X-MAC and MX-MAC can interrupt their sending streams after receiving an ACK frame.

Finally, if a node has multiple unicast destinations, (see Figure 9.3(c)), a rare case in WSNs, which tend to have only one data sink, node A calculates the appropriate t_i values for each link and sends them to the intended receivers individually. Support for multi-hop networks is introduced in Section 9.2.

9.2 t_i Control For Multi-Hop Networks

The previous section validates the principle behind t_i control for one-hop networks. In this section, we expand this work to single-branch multi-hop networks: only one data source sends packets to one data sink several hops away.

In this part of the work, the source (node 0) intends to send m^* packets to the destination (node n). Each packet travels along the same slowly changing path (*i.e.,* constant for a long period of time, corresponding to our simulation time for instance) over $h = n$ hops. Each node keeps a queue of a maximum of 100 packets.

9.2.1 Challenges Introduced By Multi-Hop Control

The introduction of several hops along a source-destination path complicates key aspects of t_i control: the delay between the beginning of a transmission at the source and its reception at the destination greatly increases. This delay is exacerbated by the nature of *LPL* MAC protocols because they rely on duty cycling. One consequence for t_i control is that instability increases, although it can be compensated by a smaller updating slope α of the command $u(t)$—we lower it to 0.1 or less.

Most importantly, the larger number of hops on the path induces non-linearities in the system. Before a packet can be transmitted, a node must wait for the packet's next-hop to wake-up. At every link along the path, the packet is held for a varying amount of time (although on average equal to ${}^{t_i}/_2$ s). Since the duty cycle is usually reevaluated every 5 to 10 packets, the packet delay (and its corollary, the number of transmitted packets m) show wide variations from one feedback to the next, with little correlation to the t_i value.

In addition to this problem, two approaches to control the duty cycle can be considered: a per-link strategy and a per-path strategy. The former strategy offered the appeal of simply replicating the work done in Section 9.1 for every link along the path, and we tried it first. Investigative work rapidly showed that this approach could not be successful because queueing would happen at one point in the path, deceiving other nodes into increasing their t_i because they correctly transmit m^* packets. In general, this solution offered many untractable problems such as keeping a set of two values of t_i at every node (one for the node itself, one for its next-hop so that the first one could send to the second), coordinating together to avoid queuing, etc.

Instead, the simple observation was made that since only one packet may be transmitted by a node k every t_i s, the nodes farther along the path ($> k$) would witness the same packet rate. Conversely, nodes placed before k would need to send at the same rate as k in order to maintain a constant queue at k. Therefore, we opted for a common duty cycle among all the nodes of a path, avoiding queuing whenever possible.

9.2.2 Node Synchronization Along a Path

While per-path t_i control eased many of the challenges we faced, non-linearities remained the main obstacle to multi-hop duty cycle control. We solved this problem through node synchronization along a path.

We used the property of certain *LPL* MAC protocols that have the unique ability to synchronize without explicit notification (*i.e.,* without overhead) along a slowly-changing one-branch path. The synchronization process was presented in Chapter 8. A node following the schedule of either one of the *LPL-Int* protocols learns of its next-hop neighbor $k+1$'s wake-up time at the end of every unicast transmission, that is, when it receives an acknowledgement frame. It follows that a node k can decide to back-off by a small t_S time so that it may wake-up right before $k+1$ during the next cycle. Done at every node along a path of h hops, nodes are automatically synchronized after the h^{th} packet has been successfully received.

Among other features, path synchronization allows urgent packets to be received and forwarded immediately (within the same t_i period) without loss of synchrony. Broadcast packets do not break node synchronization either.

More importantly, this technique reintroduces linearity in the system since nodes' wake-up times are separated by a constant amount of time $t_S\ s$. Packet delays are equal to $t_{Rx} + t_S + (h-1)(t_i + t_S)$ for regular packets, and $ht_S + t_{Rx}$ for urgent ones, where t_{Rx} is the time to receive a packet, approximately $14\ ms$ on average for our packet size.

9.2.3 Impact on the Energy Component of J

Because the wake-up schedules of nodes are staggered, the time to transmit a packet is predictable and almost constant ($t_S + t_{Rx}$), whatever the duration of t_i. In addition, since there is only one data source per path, each relay node must receive and send the same number of packets. Therefore, the expected energy consumption is the same at every relay node along the path since both the energies to send and receive a packet are equal at every hop. This reinforces the decision to utilize path-long duty-cycles.

In the case of only one data source on the network, saving the energy of one particular node on a multi-hop path no longer applies since all relays are expected to consume the same energy. Consequently, to lower the energy consumption of every relay node, the number of probes must be lowered such that a node may only wake-up to send or receive a packet. The data source or the data sink are notable exceptions, since the originator of the data does not receive packets. The data sink, which does not send packets, is generally a node with larger resources and is less likely to request its energy be spared. Unless specified otherwise, we discuss the more general results of the relay nodes, although similar techniques can be applied for the nodes at the extremities of the path, as is done in Section 9.1.

Along synchronized paths, the energy consumption is thus the lowest when t_i is the highest but still allows the target number of packets m^* to be received. Hence, the controller arbitrates the trade-off between lower energy consumption and the objective to send m^* packets.

9.2.4 Practical Considerations

The control of the duty cycle requires information now located more than one-hop away. In this section, we discuss possible practical solutions for implementing multi-hop duty cycle control.

9.2.4.1 Target Number of Packets

In all cases, we set packets to the same high priority. This meant that for path synchronization, they were all treated as urgent, and could thus be delivered within the same t_i period ($ht_S + t_{Rx}$ s later). In this section, we discuss where to close the feedback loop, *i.e.,* which node should be the t_i controller.

9.2.4.1.1 Calculation at Node 0 The target number of packets m^* now depends on the packet rate of the data source, located at the beginning of a multi-hop path. To calculate the new t_i value every feedback period $\mathcal{T}$, a node must know m^*, as well as the actual number of packets m received by the destination n. Because all nodes are sharing the same t_i value and because they are synchronized along the path, the number of packets sent by node 0 is equal to m, provided none of the packets are dropped for unforseen reasons (a bad radio state, localized noise spike, etc.).

Once calculated at node 0, the new t_i value can then be piggy-backed onto a data packet. As it is propagated along the path, a relay node adopts the new t_i value after the packet has been forwarded to its next-hop child node to ensure proper delivery of the packet.

9.2.4.1.2 Calculation at Node $n - 1$ The previous technique does not guarantee proper delivery of m^* packets at the destination if some of the links along the path are faulty. Because node $n - 1$ receives an ACK frame every time the destination receives a packet, it can easily calculate m. For this reason, the next-to-last node can be chosen to perform t_i control.

In this case, node 0 must inform its parent nodes of its intended data rate only once until the next change. It may do so by piggy-backing a byte to a unicast packet, which will be read by all the relay nodes on the path. The new t_i value must be broadcast and flooded onto the path.

This method may be preferred by programmers who suspect that nodes may fail and that detection of such failures will be slow. However, there is an inherent trade-off between packet overhead to spread the new t_i value and delivery reliability.

9.2.5 Simulation Results

We used Matlab to simulate these different strategies for a four-hop network with only one source. The control of the duty cycle was strikingly similar, whether t_i was calculated at node 0 or node $n-1$. However, our simulation did not model unforseen congestion at nodes > 1 (caused by other transmissions in the vicinity of a node for instance), thus allowing the controller at node 0 to perform equally well.

Since we later present results that were obtained when node $n-1$ does t_i control (Section 9.3), this section will focus on results using node 0 as the t_i controller to show the similarity of the techniques.

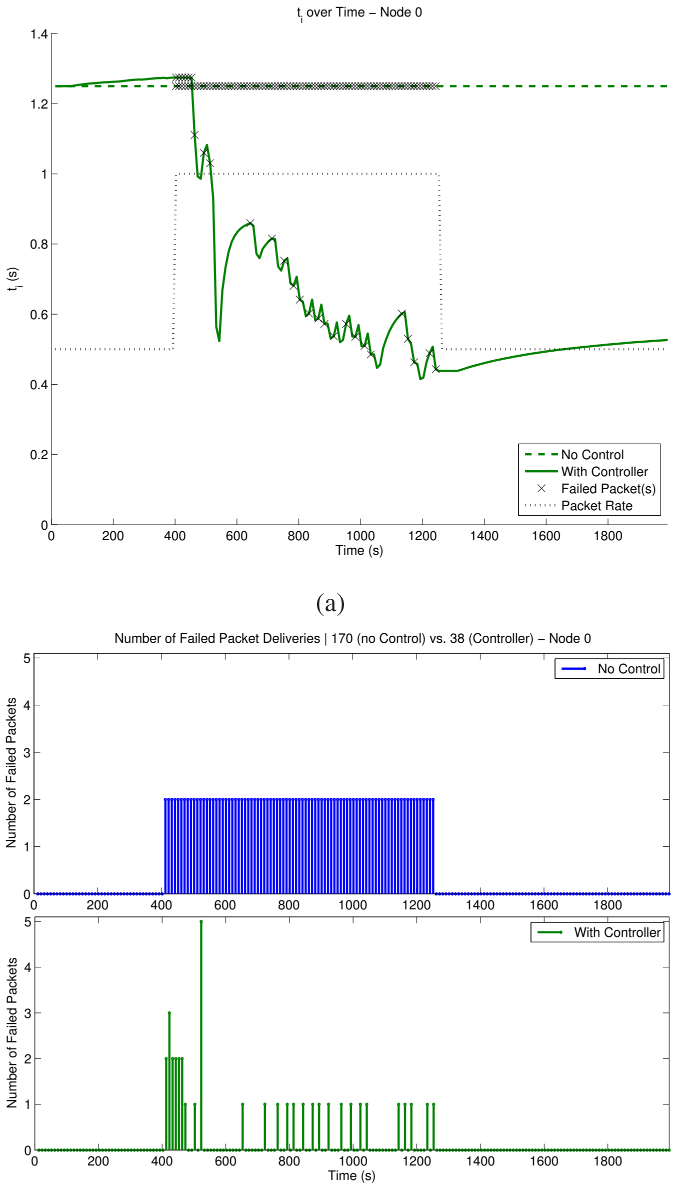
We first present results that were obtained through the direct evaluation of the consumed energy. Figure 9.4(a) shows the evolution of the duty cycle of the nodes when the packet rate of the source changes over time. When the packet rate of source node 0 doubles after $400\ s$, it fails to send m^* packets per $\mathcal{T}$ period. The controller is successful in bringing the duty cycle to a value that allows the target number of packets to be reached. When the packet rate returns to $0.5\ pkt.s^{-1}$, the duty cycle decreases again. Because we opted for a small update rate $\alpha = 0.1$, t_i increases slowly.

Figure 9.4(b) shows that t_i control reduces the number of packets failing to be delivered. With the controller, the number of dropped packets is cut by a factor of four. The number of packets transmitted by node 0 to node 1 are delivered to the destination within the same cycle. For the non-controlled case however, packets must be queued between $400\ s$ and $1,250\ s$. While queued packets can be eventually sent to the destination after the packet rate decreases, stale information is of little use to the application.

This improvement in packet delivery is obtained by a relative increase in energy consumption of 10% at relay nodes after $2,000\ s$, although the eventual energy consumption at the relay node can be eased when the duty cycle returns to a low value.

The second set of results is presented for the case when the energy consumption is not evaluated, and the command $u(t)$ is automatically increased by $0.1\ s$ when $5\ \mathcal{T}$ have passed without packet loss.

Figure 9.5(a) shows the correct reduction of t_i to accommodate sending more packets. When the network is favorable to a t_i increase, the response is more sluggish than in the regular case. However, this translates in fewer dropped packets (reduced by a factor of 9 in Figure 9.5(b)), but in a higher energy consumption of 14% over the non-



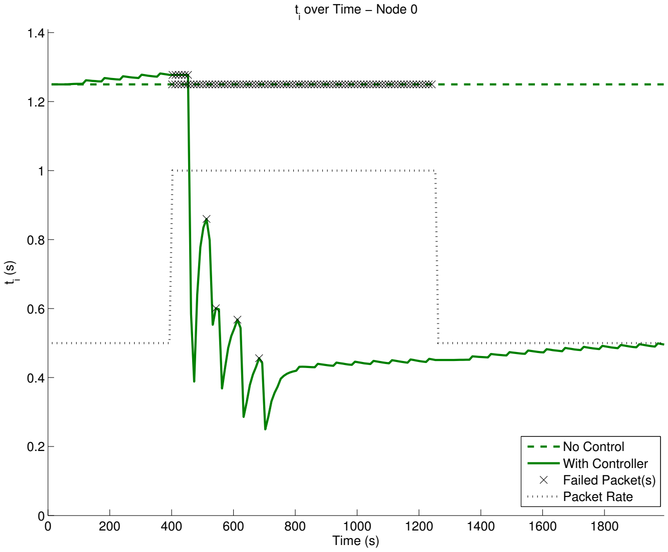
(a)

(b)

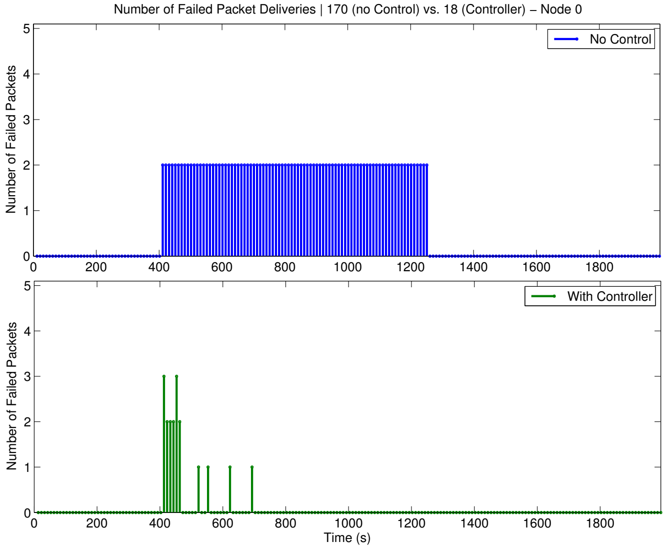
Figure 9.4: Comparison of (a) the evolution of t_i and (b) the dropped packets for the controlled and non-controlled cases when the energy is evaluated.

controlled network.

These results illustrate the trade-off existing between the two techniques for utilizing energy information, described in Section 9.1.3.2: the speed of the response translates into different energy consumption and packet delivery ratios. The decision to



(a)



(b)

Figure 9.5: Comparison of (a) the evolution of t_i and (b) the dropped packets for the controlled and non-controlled cases when the energy is not evaluated as per Section 9.1.3.2.

implement one technique or the other depends on the application needs and constraints.

9.3 t_i Control for Multi-Hop Networks With Multiple Sources

While adaptation of our scheme to multi-hop networks has greatly expanded the applications of t_i control, the limitation imposed by only one data source limits its use to networks performing source selection—target tracking or building monitoring networks are instances of these networks.

In this section, we modify our t_i control strategy to support multi-hop networks with multiple data sources.

9.3.1 t_i Control For Synchronized Paths With Multiple Sources

The greatest challenge posed by the use of multiple sources does not directly fall onto the theory behind t_i control, but rather concerns how path synchronization can be maintained when several sources are converging at one node.

The key idea to support the convergence of l flows to one node k is to use the path synchronization for multiple sources presented in Section 8.3. While t_i control does not need path synchronization for multiple branches *per se*, the synchronization technique allows per-flow fairness and avoids wasting energy on the branches. Occasional denial of service, as observed in Figure 8.9(b) would push the duty cycle beyond the real needs of the network. Compared to that approach, path synchronization results in energy savings for the branches of the network since their t_i value can be double that of the root (for two sources).

Since the node controlling the duty cycle must learn about the target number of packets m^*, it should be placed after the branch node. Thus, node $n-1$ is a good candidate to be the controller for a path with multiple branches. Upon starting and stopping its flow of packets, a source must notify the controller of the number of packets it needs to transmit every second. This value is piggy-backed onto the unicast data packet and is read by the controller. After the new t_i value has been computed, it is broadcast and flooded onto the path.

9.3.2 Simulation Results

The tested network consists of two sources sending packets over a three-hop path with initial t_i of $1.25\ s$. The controller is the node placed before the destination, and this is compared to the case when the duty cycle is fixed. However, both schemes benefit from path synchronization to guarantee fairness in the comparison.

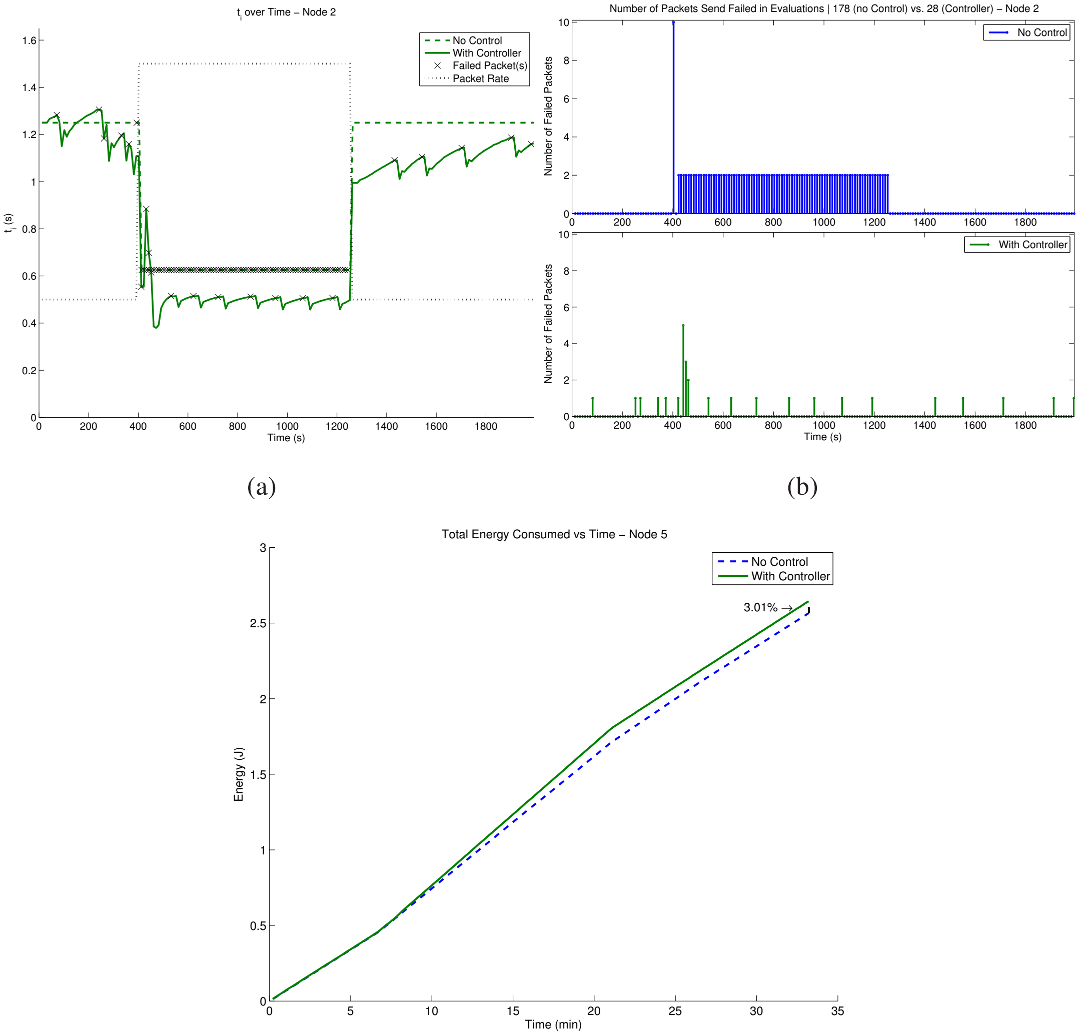
Figure 9.6(a) shows the evolution of the duty cycle for the controlled and non-controlled networks. During the initial phase of the simulation, the network experiences difficulties delivering all its packets, and thus decreases t_i. After $400\ s$, the second source is turned on, and the total packet rate is tripled. The branch synchronization process activates, which can be visualized by a division of t_i by two. The duty cycle of the network without t_i control is still halved because of the synchronization process taking place after a second source has been detected. The response from the controller is to correctly raise the duty cycle to accommodate the new load. During this time, the controller is a little too eager to increase the t_i value, and occasionally exceeds a safe value (around $500\ ms$) that allows delivery of m^* packets. At $1,250\ s$, the second source is turned back off, and the duty cycle is reduced to save energy.

The controlled network was able to drop fewer packets by a factor of six compared to the non-controlled network: Figure 9.6(b) shows that packets are mostly lost after the second source is turned on. Immediately following its activation (after $400\ s$), a loss of three, then two packets pushes the t_i value lower.

Figure 9.6(c) shows the extra energy consumed at node 2 when the duty cycle is increased. The controller's decisions to reduce t_i raises energy consumption by 3%. There is a clear trade-off between improvement in quality of service (through the increase in immediate delivery of packets) and energy consumption. However, it can be argued that although the energy expanded in the case of the non-controlled scheme is lower (because of a lower duty cycle and because contention forces nodes to sleep longer), it is done in vain since many of the packets fail to be delivered.

9.4 Summary

Low-Power-Listening MAC protocols show great promise to increase WSN lifetime by reducing idle listening. However, such MAC protocols were typically reserved for



(a) (b)

(c)

Figure 9.6: Comparison of (a) the evolution of t_i, (b) the dropped packets and (c) the energy consumed for the controlled and non-controlled cases for two sources in a multi-hop network.

networks with low packet rates so as to allow low duty cycles (and greater energy savings).

In this chapter, we introduce a control theory method that jointly optimizes the energy consumed at vulnerable nodes, and the number of packets to be transmitted over one-hop networks. This results in energy savings on the order of 20%, or in a drastic reduction of dropped packets.

We generalized these results to multi-hop networks with multiple sources. The key to successful t_i control was a path synchronization technique that allowed linearity to be maintained in the system. In our experiment, we saw a reduction of dropped packets by a factor of six when the offered load increases. The higher duty cycle caused only a limited increase in energy consumption.

This work showed that t_i control allows networks to respond to sudden bursts of packets as caused by the occurrence of an event, making *LPL* MAC protocols fit for a greater number of WSN applications. t_i control allows network designers to choose a very low duty cycle, thus saving considerable amounts of energy when the network load is low, while accommodating higher loads whenever needed. The increase in delivered packets typically comes at a temporary higher premium on energy consumption, although energy savings mean little if the network is unable to serve the application.

More importantly, the proposed t_i control method, which does not require knowledge of a system's physical model, can also be applied to the control of many other parameters in a network.

This method of controlling t_i may be compared to the TCP / IP congestion control technique. Other estimators may also be considered for our work: we could replace the NLMS algorithm by the Newton's method of gradient descent.

This chapter ends our investigation of some of the possibilities for *LPL* MAC protocol adaptation and optimization offered by X-Lisa and the Middleware Interpreter. We conclude this thesis in the next chapter.

Chapter 10

Conclusions and Future Directions

This thesis suggests ways to adapt sensor networks to conditions in the network and to application requirements by exploiting and organizing cross-layer interactions. To this end, we propose a new cross-layer information sharing architecture with a Middleware Interpreter as well as ways in which protocols at various levels in the stack can be adapted to exploit this network and application information.

10.1 Summary of Contributions

This dissertation compiles the work that has been completed toward adaptability in wireless sensor networks and includes the following:

- A case study that suggests that the majority of the gain from cross-layer designs stems from information exchange among the layers;
- Based on this observation, we proposed the X-Lisa architecture, which standardizes cross-layer information exchange to facilitate flexibility while providing opportunity for protocol adaptation;
- We introduced the *Middleware Interpreter*, which channels information between middleware and the protocol stack and allows proactive query notification;
- The application provides important additional information to protocols. Our research showed how middleware helps tune cross-layer "knobs" to improve the quality of service provided to the end user;

- We showed how the availability of information provided by X-Lisa allows MAC protocols to adapt their schedules to information about the packet and the network, thereby improving their performance;
- Such protocols can also adapt their wake-up schedule and duty-cycle to lower energy consumption and delivery delay, and increase packet delivery ratio in changing network conditions.

10.2 Future Directions

Future research will keep the objective of adapting the sensor nodes' behavior to local conditions in the network and application needs at all time. Consequently, we plan to investigate further how control theory results may be applied to other aspects in the network such as flow fairness and load balancing.

We plan to refine X-Lisa by standardizing the information exchange between adjacent layers. Right now, even with X-Lisa, programmers may choose to design complex cross-layer interactions between two neighboring layers. However, this may hinder software development and maintenance. A fixed interface should instead be provided between layers that allows receiving and sending packets, as well as requesting buffers and path information.

We will work to make the Middleware Interpreter even more expressive by allowing to define logical connectors between subqueries to form a composite query. Currently, all subqueries must have fired in order to trigger a composite query (logical "AND"); we will investigate how to specify different logical functions such as OR, XOR, etc.

We would like to propose a new way of analyzing protocols in terms of energy consumption and delay. It follows that we can define an application cost that may be modulated as a function of delay and energy consumption objectives, which optimizes for both metrics concurrently. As conditions in the network change and application requirements evolve, the routing protocol may thus be able to favor routes that are more energy efficient or that induce smaller packet delivery delays. The decision to select one path over another one can originate from information channeled by the Middleware Interpreter.

We also plan to introduce results from neural network theory: as applications for sensor networks are becoming more involved, sensor nodes will be asked to recognize very complex and noisy patterns. Multi-layer neural networks offer guidelines to train networks to identify very fuzzy patterns. Another type of neural network will allow formation of self-organizing groups in a sensor network, leading to, among other applications, in-network sensor troubleshooting and even live social networking.

Bibliography

[1] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for network sensors," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[2] University of California, Berkeley. TinyOS: http://www.tinyos.net.

[3] D. White, E. Arseneau, and C. Cifuentes, "Squawk: A Java VM for wireless sensor and actuator networks." http://developers.sun.com/learning/javaoneonline/2006/coolstuff/TS-1598.pdf?, 2006. Sun Microsystems Inc.

[4] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," in *ACM/Kluwer Mobile Networks and Applications (MONET), Special Issue on Wireless Sensor Networks, guest co-editors P. Ramanathan, R. Govindan and K. Sivalingam*, vol. 10, pp. 563–579, Aug. 2005.

[5] P. Compston, M. Styles, and S. Kalyanasundaram, "Low energy impact damage modes in aluminum foam and polymer foam sandwich structures," in *Journal of Sandwich Structures and Materials*, vol. 8, Sept. 2006.

[6] M. Perillo and W. Heinzelman, "DAPR: A protocol for wireless sensor networks utilizing an application-based routing cost," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'04)*, 2004.

[7] A. G. Ruzzelli, G. M. P. O'Hare, M. J. O'Grady, and R. Tynan, "MERLIN: A synergetic integration of MAC and routing protocol for distributed sensor networks," in *Proceedings of the Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)*, Sept. 2006.

[8] I. Rhee and J. Lee, "Energy-efficient route-aware MAC protocols for diffusion-based sensor networks," in *NCSU Technical Report 2004-4-13, CSC, NCSU*, 2005.

[9] F. Ye, G. Zhong, J. Cheng, S. Lu, and L. Zhang, "PEAS: A robust energy conserving protocol for long-lived sensor networks," in *Proceedings of the Twenty-Third International Conference on Distributed Computing Systems (ICDCS)*, 2003.

[10] C. Gui and P. Mohapatra, "Power conservation and quality of surveillance in target tracking sensor networks," in *Proceedings of the* 10^{th} *Annual International Conference on Mobile Computing and Networking (ACM MobiCom)*, Oct. 2004.

[11] F. Ye, G. Z. abd S. Lu, and L. Zhang, "A robust data delivery protocol for large scale sensor networks," in *Proceedings of the Second International Symposium on Information Processing in Sensor Networks (IPSN)*, 2003.

[12] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad-hoc wireless networks," in *Proceedings 3rd ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications DIAL M99*, pp. 48–55, Aug. 1999.

[13] B. Karp and H. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, 2000.

[14] C. E. Perkins and E. M. Royer, "Ad hoc on-demand distance vector routing," in *Proceedings of the* 2^{nd} *IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, Feb. 1999.

[15] K. Langendoen, "Medium access control in wireless sensor networks," in *Medium Access Control in Wireless Networks, Volume II: Practice and Standards*, Nova Science Publishers, 2007.

[16] A. El-Hoiydi, “Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks,” in *Proceedings of the IEEE International Conference on Communications (ICC’02)*, Apr. 2002.

[17] A. El-Hoiydi and J. Decotignie, “WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks,” in *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, July 2004.

[18] J. Polastre, J. Hill, and D. Culler, “Versatile low power media access for wireless sensor networks,” in *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys’04)*, pp. 95–107, Nov. 2004.

[19] Y. Wei, J. Heidemann, and D. Estrin, “An energy-efficient mac protocol for wireless sensor networks,” in *Proceedings of the Twenty-First International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’02)*, June 2002.

[20] T. van Dam and K. Langendoen, “An adaptive energy-efficient mac protocol for wireless sensor networks,” in *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys’03)*, Oct. 2003.

[21] M. Buettner, G. V. Yee, E. Anderson, and R. Han, “X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks,” in *Proceedings of the 4th Embedded Networked Sensor Systems (SenSys’06)*, pp. 307–320, Nov. 2006.

[22] K.-J. Wong and D. Arvind, “SpeckMAC: Low-power decentralised MAC protocol low data rate transmissions in Specknets,” in *Proceedings 2nd IEEE International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN’06)*, May 2006.

[23] S. Liu, K.-W. Fan, and P. Sinha, “CMAC: An energy efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks,” in *Proceedings of the Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON’07)*, June 2007.

[24] G. Lu, B. Krishnamachari, and C. Raghavendra, “An adaptive energy-efficient and low-latency MAC for data gathering in sensor networks,” in *Proceedings of the Fourth International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, 2004.

[25] A. Keshavarzian, H. Lee, and L. Venkatraman, “Wakeup scheduling in wireless sensor networks,” in *Proceedings of the Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc’06)*, May 2006.

[26] V. Srivastava and M. Motani, “Cross-layer design: A survey and the road ahead,” in *IEEE Communications Magazine*, vol. 43, pp. 112–119, Dec. 2005.

[27] V. Kawadia and P. Kumar, “A cautionary perspective on cross-layer design,” in *Proceedings Wireless Communications*, Feb. 2005.

[28] L. V. Hoesel, T. Nieberg, J. Wu, and P. J. M. Havinga, “Prolonging the lifetime of wireless sensor networks by cross-layer interaction,” in *IEEE Wireless Communications*, pp. 78–86, Dec. 2004.

[29] X. Lin and N. Shroff, “The impact of imperfect scheduling on cross-layer rate control in wireless networks,” in *Proceedings of 24th International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’05)*, Mar. 2005.

[30] B. Kim, Y. Fang, and T. Wong, “Rate-adaptive MAC protocol in high-rate personal area networks,” in *Proceedings IEEE Wireless Communication and Networking Conference (WCNC)*, Mar. 2004.

[31] H. Pham and S. Jha, “An adaptive mobility-aware MAC protocol for sensor networks (MS-MAC),” in *Proceedings 1st Conference on Mobile Ad-hoc and Sensor Systems (MASS’04)*, Oct. 2004.

[32] R. Jurdak, P. Baldi, and C. V. Lopes, “Adaptive low power listening for wireless sensor networks,” in *IEEE Transactions on Mobile Computing*, vol. 6, Aug. 2007.

[33] C. M. Vigorito, D. Ganesan, and A. G. Barto, “Adaptive control of duty cycling in energy-harvesting wireless sensor networks,” in *Proceedings of The Fourth IEEE*

Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'07), June 2007.

[34] H. K. Le, D. Henriksson, and T. Abdelzaher, "A control theory approach to throughput optimization in multi-channel collection sensor networks," in *Proceedings of the Sixth International Symposium on Information Processing in Sensor Networks (IPSN'07)*, Apr. 2007.

[35] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "Hood: A neighborhood abstraction for sensor networks," in *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services (MobiSYS'04)*, June 2004.

[36] Q. Wang and M. A. Abu-Rgheff, "Cross-layer signalling for next-generation wireless systems," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'03)*, vol. 2, pp. 1084–1089, Mar. 2003.

[37] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross-layering in mobile ad hoc network design," in *IEEE Computer*, pp. 48–51, Feb. 2004.

[38] C. M. Sadler, L. Kant, and W. Chen, "Cross-layer self-healing mechanisms in wireless networks," in *Proceedings* 6^{th} *World Wireless Congress (WWC'05)*, May 2005.

[39] R. Winter, J. H. Schiller, N. Nikaein, and C. Bonnet, "Crosstalk: Cross-layer decision support based on global knowledge," in *IEEE Communications Magazine*, Jan. 2006.

[40] C. T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica, "A modular network layer for sensornets," in *Proceedings* 7^{th} *Symposium on Operating Systems Design and Implementation (OSDI'06)*, Nov. 2006.

[41] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica, "A unifying link abstraction for wireless sensor networks," in *Proceedings Proceedings of the* 3^{rd} *Embedded Networked Sensor Systems (SenSys'05)*, Nov. 2005.

[42] A. Dunkels, F. Osterlind, and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proceedings* 1^{st} *ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, Nov. 2007.

[43] I. F. Akyildiz, M. C. Vuran, and Ö. B. Akan, “A cross-layer protocol for wireless sensor networks,” in *Proceedings of the Conference on Information Science and Systems (CISS’06)*, (Princeton, NJ), pp. 22–24, Mar. 2006.

[44] K. Römer, “Time synchronization in ad hoc networks,” in *Proceedings of the Second ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2001.

[45] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[46] M. Sichitiu and V. Ramadurai, “Localization of wireless sensor networks with a mobile beacon,” in *Proceedings* 1^{st} *IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS’04)*, Oct. 2004.

[47] E. Ould-Ahmed-Vall, D. Blough, B. Heck, and G. Riley, “Distributed unique global id assignment for sensor networks,” in *Proceedings* 2^{nd} *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference (MASS’05*, Nov. 2005.

[48] B. Blum, P. Nagaraddi, A. Wood, T. Abdelzaher, S. Son, and J. Stankovic, “An entity maintenance and connection service for sensor networks,” in *Proceedings The First International Conference on Mobile Systems, Applications, and Services (MobiSys’03)*, May 2003.

[49] V. Lenders, M. May, and B. Plattner, “Towards a new communication paradigm for mobile ad hoc networks,” in *Proceedings* 2^{nd} *IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS MHWMN’05 Workshop*, Oct. 2005.

[50] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, “Internet indirection infrastructure,” in *Proceedings SIGCOMM’02*, 2002.

[51] K. Römer, O. Kasten, and F. Mattern, “Middleware challenges for wireless sensor networks,” in *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, 2002.

[52] M.-M. Wang, J.-N. Cao, J. Li, and S. K. Das, "Middleware for wireless sensor networks: A survey," in *Journal of Computer Science and Technology*, vol. 23, pp. 305–326, May 2008.

[53] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," in *IEEE Network*, vol. 18, pp. 6–14, Jan. 2004.

[54] S. Li, Y. Lin, S. H. Son, J. A. Stankovic, and Y. Wei, "Event detection services using data service middleware in distributed sensor networks," in *Proceedings of the* 2^{nd} *International Conference on Information Processing in Sensor Networks*, pp. 502–517, Apr. 2003.

[55] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. D. Kim, B. Zhou, and E. G. Sirer, "On the need for system-level support for ad hoc and sensor networks," in *Operating Systems Review*, vol. 36, 2002.

[56] Q. Han and N. Venkatasubramanian, "AutoSeC: An integrated middleware framework for dynamic service brokering," in *IEEE Distributed Systems Online*, vol. 2, 2001.

[57] T. Liu and M. Martonosi, "Impala: A middleware system for managing autonomic, parallel sensor systems," in *Proceedings of the* 9^{th} *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03)*, pp. 107–118, June 2003.

[58] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," in *IEEE Network Magazine*, vol. 18, Jan. 2004.

[59] R. Cornea, S. Mohapatra, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Managing cross-layer constraints for interactive mobile multimedia," in *Proceedings of the IEEE Workshop on Constraint-Aware Embedded Software*, 2003.

[60] "Network simulator ns-2.28 and ns 2-1b." http://www.isi.edu/nsnam/ns/.

[61] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, pp. 660–670, Oct. 2002.

[62] B. Hamdaoui and P. Ramanathan, “Lifetime-throughput tradeoff for elastic traffic in multi-hop hotspot networks,” in *Proceedings* 47^{th} *Annual IEEE Global Telecommunications Conference (Globecom’04)*, Dec. 2004.

[63] W. Su and T. Lim, “Cross-layer design and optimization for wireless sensor networks,” in 7^{th} *ACIS International Conference on Soft. Engr., Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD’06)*, pp. 278–284, June 2006.

[64] C. J. Merlin and W. B. Heinzelman, “Cross-layer gains for sensor networks,” in *Proceedings Conference on Distributed Computing in Sensor Systems (DCOSS’06) Poster Session*, June 2006.

[65] H. Zimmermann, “OSI reference model—the ISO model of architecture for open systems interconnection,” in *IEEE Transactions on Communications*, vol. 28, pp. 425–432, Apr. 1980.

[66] A. Belenki, Product Director, Luxoft Labs, “Overcoming challenges of TinyOS use in commercial ZigBee applications,” in *TinyOS Technology Exchange III*, Feb. 2006.

[67] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, “Geographic routing without location information,” in *Proceedings of the Ninth Annual International Conference on Mobile Computing and Networking (MobiCom’03)*, 2003.

[68] MoteIV Tmote sky. http://www.moteiv.com/tmote.

[69] C. J. Merlin and W. B. Heinzelman, “Sensor network middleware for managing a cross-layer architecture,” in *Proceedings DCOSS’06 - EAWMS Workshop*, June 2006.

[70] Philip Lewis et al., “Ad-hoc routing component architecture.” http://www.tinyos.net/tinyos-1.x/doc/ad-hoc.pdf, 2003.

[71] T. Stathopoulos, L. Girod, J. Heidemann, and D. Estrin, “Mote herding for tiered wireless sensor networks,” in *CENS Tech. Rep. #58*, Dec. 2005.

[72] B. Otal and L. Alonso, “A cross-layer energy-saving mechanism for an enhancement of 802.11 WLAN systems,” in *Proceedings of Vehicular Technology Conference (VTC’04)*, May 2004.

[73] K. Arisha, M. Youssef, and M. Younis, “Energy-aware TDMA based MAC for sensor networks,” in *IEEE Workshop on Integrated Management of Power Aware Communications Computing and Networking (IMPACCT’02)*, 2002.

[74] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, “TAG: a tiny aggregation service for ad-hoc sensor networks,” in *Proceedings of the ACM Symposium on Operating System Design and Implementation (OSDI’02)*, 2002.

[75] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz, “A message-oriented middleware for sensor networks,” in *Proceedings of the* 2^{nd} *Workshop on Middleware for Pervasive and Ad-hoc Computing (MPAC’04)*, pp. 127–134, Oct. 2004.

[76] O. Yang, C. J. Merlin, and W. B. Heinzelman, “A general cost function to reflect sensor support for application requirements,” in *Proceedings of the* 4^{th} *Conference on Distributed Computing in Sensor Systems (DCOSS’07 Poster Session)*, June 2007.

[77] X. Shi and G. Stromberg, “SyncWUF: An ultra low-power mac protocol for wireless sensor networks,” in *IEEE Transactions on Mobile Computing*, vol. 6, pp. 115 – 125, Jan. 2007.

[78] S. Mahlknecht and M. Böck, “CSMA-MPS: A minimum preamble sampling mac protocol for low power wireless sensor networks,” in *Proceedings of the IEEE International Workshop on Factory Communication Systems*, Sept. 2004.

[79] G. C. Goodwin and K. S. Sin in *Adaptive Filtering Prediction and Control*, Prentice-Hall, 1984.

[80] P. Kumar and P. Varaiya in *Stochastic Systems: Estimation, Identification, and Adaptive Control*, Prentice-Hall, 1986.