RESILIENT SENSOR NETWORK
QUERY PROCESSING

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2014

By
Alan Barry Stokes
School of Computer Science
Contents

Abstract 24

Publications 26

Declaration 27

Copyright 28

Acknowledgement 30

1 Introduction 31
   1.1 Setting the Scene ................................. 31
   1.2 Motivation ........................................ 32
   1.3 Sensor Network Query Processors ...................... 33
   1.4 Aim, Objectives, and Contributions .................... 40
   1.5 Overview of Thesis Structure ........................ 42

2 Background 44
   2.1 Query Processing ...................................... 45
      2.1.1 Classical Query Processing ....................... 45
      2.1.2 Distributed Query Processing .................... 46
      2.1.3 Stream Query Processing ........................ 50
2.2 Wireless Sensor Networks ........................................ 52
   2.2.1 Deployments ............................................. 52
   2.2.2 Quality of Service in Wireless Sensor Networks ....... 54

2.3 SNEE SNQP ..................................................... 58
   2.3.1 SNEE overview ........................................... 59
   2.3.2 SNEE Inputs ............................................. 60
   2.3.3 SNEE single site optimisation phase .................. 65
   2.3.4 SNEE Multi-Site Optimisation Phase ................. 65
      2.3.4.1 Routing ........................................... 66
      2.3.4.2 Where-Scheduling .................................. 68
      2.3.4.3 When-Scheduling .................................. 75
      2.3.4.4 Code Generation .................................. 77

2.4 Summary ....................................................... 80

3 Resilience to Unpredictable Node Events ..................... 82
   3.1 Overview ................................................... 84
      3.1.1 Monitoring Technique ................................ 86
   3.2 Complete Re-optimisation ................................ 88
      3.2.1 Motivation ............................................ 89
      3.2.2 Execution Stack ...................................... 89
      3.2.3 Adaptation Break Down ................................ 90
   3.3 Regional Re-optimisation ................................ 92
      3.3.1 Motivation ............................................ 93
      3.3.2 Execution stack ...................................... 93
         3.3.2.1 Node Pinning .................................... 94
         3.3.2.2 Routing and Unpinning ........................... 95
         3.3.2.3 Where Scheduling ................................. 98
4.2.2.2 Generating the Initial Population .......................... 151
4.2.2.3 Mutation and Crossover ................................... 151
4.2.2.4 Stopping Criteria ......................................... 152
4.2.3 Run Time Module ............................................ 153
4.3 Evaluation ....................................................... 154
4.3.1 Experimental Design ........................................ 155
4.3.2 Results ....................................................... 158
4.3.2.1 Stable Deployments: Lifetime Measurements ....... 158
4.3.2.2 Unstable Deployments Lifetime Measurements ... 164
4.3.2.3 Robustness Measurements ............................... 167
4.3.2.4 Adaptation Time Measurements ........................ 170
4.3.2.5 Switch Time Suitability ................................ 172
4.4 Discussion ....................................................... 175
4.5 Related Work .................................................... 176
4.5.1 Routing Based Techniques ................................. 177
4.5.2 Plan Based Techniques ...................................... 178
4.5.3 Redundancy-Based Techniques .............................. 178
4.6 Conclusion ....................................................... 179

5 Resiliently Designed Deployments .................................. 181
5.1 Motivation ....................................................... 181
5.2 K-Resilient QEPs ................................................ 182
5.2.1 Modifying a Deployment for Planned Redundancy .... 185
5.3 Resilience Through Logical Nodes ............................... 185
5.3.1 The Generation of Super-Overlay .......................... 187
5.3.2 Generating Logical Overlays ............................... 189
5.3.3 Assessing The Logical Overlay Set ....................... 192
5.3.4 Replication of Site Tasks ........................................ 192
5.3.5 Adapting to Unpredictable Node Failure Events ........ 193
5.4 Resilience Through Logical Channels ....................... 194
  5.4.1 Motivation ...................................................... 196
  5.4.2 The Communication Protocol ............................... 197
    5.4.2.1 Modifying a QEP Agenda ............................. 197
    5.4.2.2 Runtime Tasks ......................................... 203
5.5 Evaluation ........................................................ 212
  5.5.1 Topology Generation ....................................... 213
  5.5.2 Experimental Results: Unpredictable Node Failure Events 215
    5.5.2.1 Lifetime Measurements in Unstable Deployments 216
    5.5.2.2 Adaptation Energy and Time Measurements .... 221
  5.5.3 Experimental Results: Unpredictable Communication Chan-
    nel Failure Events ............................................... 223
    5.5.3.1 Number of Tuples Delivered to the End User ........ 226
    5.5.3.2 The Distance Effect .................................... 231
5.5.4 Robustness Measurements ................................... 236
  5.5.4.1 Lifetime vs Number of Tuples in Stable Deployments 239
  5.5.4.2 Lifetime in Unstable Deployments .................... 246
5.6 Related Work .................................................... 251
  5.6.1 Predicting Communication Channel Failure Events ...... 252
  5.6.2 Reacting to Communication Channel Failure Events .... 253
5.7 Conclusions ...................................................... 254

6 Conclusions ......................................................... 256
  6.1 Research Contributions ...................................... 256
  6.2 Significance of the Results .................................. 258
B  Cardinality Model Evaluation 351

C  Single Node Failure: Supplementary Material 414

D  Robustness Measurements: Supplementary Material 430

E  Noise Model: Supplementary Material 432
   E.1 Noise Model .................................................. 432

F  SNEEq Algebra: Supplementary Material 436

G  Experimental Flows: Supplementary Material 441
   G.1 Unpredictable Several Node Failure ................. 441
      G.1.1 Lifetime Experiment Flow Diagram ............. 441
      G.1.2 Time Experiment Flow Diagram ................. 443
      G.1.3 Energy Cost Experiment Flow Diagram ......... 448
      G.1.4 Tuple Experiment Flow Diagram ................. 451
   G.2 Unpredictable Single Node Failure ................. 454
      G.2.1 Lifetime Experiment Flow Diagram ............. 454
      G.2.2 Time Experiment Flow Diagram ................. 457
      G.2.3 Energy Cost Experiment Flow Diagram ......... 459
      G.2.4 Tuple Experiment Flow Diagram ................. 461
   G.3 Sequence of Planned Transitions .................. 464
      G.3.1 Stable Lifetime Experiment Flow Diagram .... 464
      G.3.2 Unstable Lifetime Experiment Flow Diagram ... 467
      G.3.3 Suitability of Transition Times Experiment Flow Diagram 470
   G.4 Resilient Deployments ................................. 473
      G.4.1 Several Node Failure Lifetime Experiment Flow Diagram 473
      G.4.2 Adaptation Energy Cost Experiment Flow Diagram . 477
B Experimental Edge lifetime to Tuple relationship: Supplementary Material

Bibliography

Word Count: 74,307
List of Tables

1.1 Characteristics of existing SNQPs. 36

2.1 Logical query operators. 45

2.2 Characteristics of existing sensor network deployments as collated in [31]. 57

2.3 SNEEqI Physical Algebra. 62

3.1 QoSs expectations used. 101

3.2 Average and standard deviation for each query type. 112

4.1 Average improvement between proactive and the other two strategies for the different query types for QoS set 1 on the small topology set. 159

4.2 Average improvement between proactive and the other strategies for the different query types for QoS set 2 on the small topology set. 160

4.3 Average improvement between proactive and the other strategies for the different query types for QoS set 1 on the large topology set. 163

4.4 Improvement between proactive and the other strategies for the different query types on the sniper deployment. 164

4.5 Average and standard deviation for each query type. 167

4.6 Lifetimes given different switch times between successive QEPs. 172
5.1 Average improvement between RSNEE with acquisition nodes and the other two strategies for the different query types over several unexpected node failure events. ......................... 217
5.2 Average improvement between the RSNEE without acquisition nodes and the other two strategies for the different query types over several unexpected node failure events. ......................... 217
5.3 Path loss exponents for different environments [95] ......................... 224
5.4 Different set-up parameters for the hardest and easiest set-ups. ........ 224
5.5 Average performance increase for different levels of $k_e$ against the static strategy for the easy set-up ......................... 226
5.6 Average performance increase for different levels of $k_e$ against the static strategy for the hard set-up ......................... 227

B.1 table of results for topology 0 ........................................ 352
B.2 table of results for topology 1 ........................................ 353
B.3 table of results for topology 2 ........................................ 354
B.4 table of results for topology 3 ........................................ 355
B.5 table of results for topology 4 ........................................ 356
B.6 table of results for topology 5 ........................................ 358
B.7 table of results for topology 6 ........................................ 359
B.8 table of results for topology 7 ........................................ 361
B.9 table of results for topology 8 ........................................ 363
B.10 table of results for topology 9 ....................................... 365
B.11 table of results for topology 10 ..................................... 366
B.12 table of results for topology 11 ..................................... 367
B.13 table of results for topology 12 ..................................... 369
B.14 table of results for topology 13 ..................................... 371
List of Figures

1.1 Examples of Sensor Motes. ................................................. 32
1.2 Taxonomy of Related Works. .............................................. 40

2.1 An example query at different stages of classical query processing. 47
2.2 The example query in distributed form. .................................. 49
2.3 The SNEE Infrastructure. .................................................. 60
2.4 Examples of SNEE inputs. .................................................. 63
2.5 The SNEE compilation/optimisation stack [110]. .......................... 64
2.6 Query forms in the stages 1-3 of the SNEE compilation/optimisation
stack. .............................................................................. 66
2.7 An example routing tree. ...................................................... 67
2.8 An Algorithm for Computing a Routing Tree. .............................. 68
2.9 The partitioning algorithm. .................................................. 69
2.10 The fragment allocation algorithm. ....................................... 71
2.11 Query in distributed form. .................................................. 73
2.12 Query in instance fragment tree form. .................................... 74
2.13 Computing an agenda buffering factor. ................................... 76
2.14 The agenda construction algorithm. ...................................... 78
2.15 Representation of the query’s agenda. .................................... 79
2.16 TinyOS component diagram for site 8 in Figure 2.12. ................. 79
3.1 The Infrastructure of ADSNEE. ........................................... 85
3.2 Pseudo code for determining which node has experienced a *unpredictable node failure event.* ........................................... 87
3.3 Complete re-optimisation execution stack. ............................ 90
3.4 QEPs overlaid on the deployment illustrated in Figure 2.4(a). .... 91
3.5 Regional execution stack. .................................................. 95
3.6 Representation of the original and incomplete RTs. ................. 96
3.7 Representation of the different scoped incomplete and repaired RT's. 97
3.8 Representation of the original and regional generated QEP's. .... 98
3.9 Estimated lifetime for all three strategies for the select * query type over several *unexpected node failure events.* ................. 104
3.10 Estimated lifetime for all three strategies for an aggregation query type over several *unexpected node failure events.* ................. 105
3.11 Estimated lifetime for all three strategies for the join query type over several *unexpected node failure events.* ...................... 106
3.12 Estimated lifetime of both strategies in the face of several node failures for the small topology set with QoS set 1. ................. 109
3.13 Estimated lifetime of both strategies in the face of several node failures for the large topology set with QoS set 1. ................. 109
3.14 Estimated lifetime of both strategies in the face of several node failures for the small topology set with QoS set 2. ................. 110
3.15 Estimated lifetime of both strategies in the face of several node failures for the large topology set with QoS set 2. ................. 110
3.16 The lifetime estimated for a given deployment over a collection of diverse queries. ....................................................... 113
3.17 The difference in lifetime between the different strategies over a collection of diverse queries. .......................................... 114
3.18 Estimated number of tuples delivered to the end user for the different topologies in the small topology set, given QoS expectation set one. ................................................................. 117

3.19 Estimated number of tuples delivered to the end user for the different topologies in the large topology set, given QoS expectation set one. ................................................................. 117

3.20 Estimated number of tuples delivered to the end user for the different topologies in the small topology set, given QoS expectation set two. ................................................................. 118

3.21 Estimated number of tuples delivered to the end user for the different topologies in the large topology set, given QoS expectation set two. ................................................................. 118

3.22 Estimated time period taken up adapting to several node failures for the different topologies in the small topology set, given QoS expectation set one. ......................................................... 120

3.23 Estimated time period taken up adapting to several node failures for the different topologies in the large topology set, given QoS expectation set one. ......................................................... 120

3.24 Estimated time period taken up adapting to several node failures for the different topologies in the small topology set, given QoS expectation set two. ......................................................... 121

3.25 Estimated time period taken up adapting to several node failures for the different topologies in the large topology set, given QoS expectation set two. ......................................................... 121

3.26 The tipping point for adaptations without disruption. .................. 123
3.27 Estimated energy drain upon the deployment experienced from several adaptation events for topologies in the small topology set, with QoS set 1. ................................. 125
3.28 Estimated energy drain upon the deployment experienced from several adaptation events for different topologies and queries in the large set, with QoS set 1. ................................. 125
3.29 Estimated energy drain upon the deployment experienced from several adaptation events for topologies in the small topology set, with QoS set 2. ................................. 126
3.30 Estimated energy drain upon the deployment experienced from several adaptation events for different topologies and queries in the large topology set, with QoS set 2. ................................. 126

4.1 The ADPSNEE infrastructure. ................................. 136
4.2 The strategy at a glance. ................................. 138
4.3 Computing the current best solution. ................................. 139
4.4 An example of a sequence of QEPs. ................................. 140
4.5 Step by step evolution of the tabu list. ................................. 142
4.6 Computing the neighbourhood. ................................. 144
4.7 The fitness function. ................................. 145
4.8 Computing the elite list. ................................. 150
4.9 A graphical representation of the genotype. ................................. 150
4.10 A graphical representation of the mutation and crossover. ................................. 153
4.11 Example of a sequence of QEPs given the unexpected failure of node 2. ................................. 154
4.12 A graphical representation of the sniper localisation deployment in the context of SNQP [103]. ................................. 156
4.13 Lifetimes for static, proactive, and reactive strategies for QoS set
1 for the small topology set. ........................................... 161

4.14 Lifetimes for static, proactive, and reactive strategies for QoS set
2 for the small topology set. ........................................... 161

4.15 Lifetimes for static, proactive, and reactive strategies for QoS set
1 for the large topology set. ............................................ 162

4.16 Lifetimes for static, proactive, and reactive strategies for the sniper
deployment. ............................................................. 163

4.17 Lifetimes for proactive and reactive strategies in the face of unpre-
dictable node failure for the large topology set. .................... 166

4.18 Lifetimes for proactive and reactive strategies in the face of unpre-
dictable node failure for the small topology set. .................... 166

4.19 The lifetime estimated for a given deployment over a collection of
diverse queries. ....................................................... 169

4.20 The difference in lifetime between the different strategies over a
collection of diverse queries. ........................................ 169

4.21 Time taken by switching between QEPs in a sequence. ............ 170

4.22 Comparison of the original sequence lifetime and the optimal life-
time with adjusted switch times. ................................... 173

4.23 The expected lifetime of the QEPs in each sequence generated from
the small topology set with corresponding switch times. ............ 174

5.1 The Infrastructure for RSNEE. .................................... 184

5.2 Initial and modified deployments. .................................. 186

5.3 The compilation and runtime stack for RSNEE. .................... 187

5.4 Super overlay. ....................................................... 189

5.5 A reduced logical overlay. ......................................... 190
5.6 The logical overlay network after site task replication. 192
5.7 The logical overlay network after adapting to node 8 experiencing a *unpredictable node failure event* 193
5.8 The compilation and runtime stack from the RSNEE when robust to *unpredictable communication channel failure events*. 195
5.9 A graphical representation of the ability for nodes within a logical node to communicate with each other. 196
5.10 An algorithm for computing a logical agenda. 198
5.11 An Algorithm for adding redundant tasks that together represent a logical channel. 200
5.12 Representation of the agenda after the redundant transmission tasks are added. 202
5.13 The initial set-up for the runtime protocol. 204
5.14 The runtime protocol for *SLEEPING* state. 205
5.15 The runtime protocol for *OPERATING* state. 206
5.16 The runtime protocol for *LISTENING* state. 208
5.17 The runtime protocol for *LISTENING* state when a packet is received. 209
5.18 The runtime protocol method SendAckPacketsToInputs. 210
5.19 Different transmission behaviours. 211
5.20 How to adjust the topologies from previous evaluations. 214
5.21 Estimated lifetime for all three strategies for the different query types over several *unexpected node failure events*. 220
5.22 Estimated communicative energy drain upon the deployment experienced from several adaptation events for topologies with \(k_n = 2\), and QoS set 1. 222
5.23 Estimated communicative time cost of several adaptation events for topologies with $k_n = 2$ and QoS set 1. 

5.24 Estimated number of tuples delivered to the end user over 50 agenda cycles for the select query type over different levels of $k_e$ for different set ups. 

5.25 Estimated number of tuples delivered to the end user over 50 agenda cycles for the aggregation query type over different levels of $k_e$ for different set ups. 

5.26 Estimated number of tuples delivered to the end user over 50 agenda cycles for the join query type for different levels of $k_e$ for different set ups. 

5.27 Estimated percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$. 

5.28 Estimated percentage of tuples delivered to the end user per agenda cycle for aggregation queries over different levels of $k_e$ and $k_n$. 

5.29 Estimated percentage of tuples delivered to the end user per agenda cycle for different query types, levels of $k_e$ and $k_n$ and different set ups. 

5.30 The percentage of tuples delivered per agenda cycle for a given deployment over a collection of diverse queries. 

5.31 The percentage increase of RSNEE over the static strategy in tuples delivered per agenda cycle for a given deployment over a collection of diverse queries. 

5.32 The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$. 
5.33 The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for aggregation queries over different levels of \( k_e \) and \( k_n \) ......................... 241

5.34 The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of \( k_e \) and \( k_n \) for different set ups. .................... 242

5.35 The number of tuples delivered to the end user during the lifetime of the deployment for different levels of \( k_e \) and \( k_n \) .................... 245

5.36 The lifetime of the strategies when deployed on a deployment with select query .............................. 248

5.37 The lifetime of the strategies when deployed on a deployment with aggregation query ......................... 249

5.38 The lifetime of the strategies when deployed on deployments which have experienced a number of *unpredictable node failure events* .. 250

A.1 Instance Operator Tree representing the query in Section 2.4(b) 341

B.1 topology for test 0 ........................................... 352
B.2 topology for test 1 ........................................... 353
B.3 topology for test 2 ........................................... 354
B.4 topology for test 3 ........................................... 355
B.5 topology for test 4 ........................................... 356
B.6 topology for test 5 ........................................... 357
B.7 topology for test 6 ........................................... 359
B.8 topology for test 7 ........................................... 360
B.9 topology for test 8 ........................................... 362
B.10 topology for test 9 ......................................... 364
B.11 topology for test 10 ....................................... 366

20
B.12 topology for test 11 ........................................ 367
B.13 topology for test 12 ........................................ 368
B.14 topology for test 13 ........................................ 370
B.15 topology for test 14 ........................................ 372
B.16 topology for test 15 ........................................ 373
B.17 topology for test 16 ........................................ 375
B.18 topology for test 17 ........................................ 376
B.19 topology for test 18 ........................................ 377
B.20 topology for test 19 ........................................ 379
B.21 topology for test 20 ........................................ 380
B.22 topology for test 30 ........................................ 381
B.23 IOT for test 30 ............................................. 382
B.24 topology for test 31 ........................................ 384
B.25 IOT for test 31 ............................................. 385
B.26 topology for test 32 ........................................ 387
B.27 IOT for test 32 ............................................. 388
B.28 topology for test 33 ........................................ 390
B.29 IOT for test 33 ............................................. 391
B.30 topology for test 34 ........................................ 393
B.31 IOT for test 34 ............................................. 394
B.32 topology for test 36 ........................................ 396
B.33 IOT for test 36 ............................................. 397
B.34 topology for test 60 ........................................ 399
B.35 IOT for test 60 ............................................. 400
B.36 topology for test 62 ........................................ 402
B.37 IOT for test 62 ............................................. 403
B.38 topology for test 64 ........................................ 405
B.39 IOT for test 64 ........................................ 406
B.40 topology for test 67 ................................... 408
B.41 IOT for test 67 ........................................ 409
B.42 topology for test 68 ................................... 411
B.43 IOT for test 68 ........................................ 412

C.1 Estimated lifetime of different queries and topologies with QoS set 1418
C.2 Estimated lifetime of different queries and topologies with QoS set 2418
C.3 Estimated lifetime of different queries and topologies with QoS set 3419
C.4 Estimated number of tuples delivered to the end user over different
    queries and topologies with QoS set 1 .................... 422
C.5 Estimated number of tuples delivered to the end user over different
    queries and topologies with QoS set 2 .................... 422
C.6 Estimated timer period taken up adapting to a failure for the small
    topology set with the three types of queries ............. 425
C.7 Estimated timer period taken up adapting to a failure for the large
    topology set with the three types of queries ............. 425
C.8 Estimated energy drain upon the deployment experienced from the
    adaptation process for the small topology set with the three queries 428
C.9 Estimated energy drain upon the deployment experienced from the
    adaptation process for the large topology set with the three queries 428

E.1 A typical SNR curve .................................... 433
E.2 formula for calculating SNR ................................ 433
E.3 Formula for determining received signal strength indicator (RSSI) 434
E.4 Formula for determining the signal strength lost .................. 434
E.5 Free space path loss formula ............................... 434
E.6 An illustration of how the RSSI is calculated .................. 435
E.7 formula for calculating PRR ........................................ 435

G.1 The experimental flow for the lifetime estimates in the evaluations 444
G.2 The experimental flow for the total time estimates in the evaluations447
G.3 The experimental flow for the energy used up estimates in the
evaluations ........................................................................ 450
G.4 The experimental flow for the energy used up estimates in the
evaluations ........................................................................ 453
G.5 The experimental flow for the lifetime estimates in the evaluations 456
G.6 The experimental flow for the time estimates in the evaluations . 458
G.7 The experimental flow for the energy estimates in the evaluations 460
G.8 The experimental flow for the number of tuples delivered estimates
used in the evaluations ....................................................... 463
G.9 The experimental flow for the lifetime estimates for the sequence
of planned transitions evaluations in stable environments . . . . 466
G.10 The experimental flow for the lifetime estimates for the sequence
of planned transitions evaluations in unstable environments . . . 469
G.11 The experimental flow for the suitability of the transition times of
a sequence ................................................................. 472
G.12 The experimental flow for the lifetime estimate of the resilient
strategy in the face of several node failure events. ................. 476
G.13 The experimental flow for the total energy estimate of the resilient
strategy in the face of several node failure events. ................. 480
G.14 The experimental flow for the total time estimate of the resilient
strategy in the face of several node failure events. ................. 484
G.15 The experimental flow for determining if the total number of tuples delivered to the end user in the face of unpredictable edge failure events. ................................................. 487

G.16 The experimental flow for determining the estimated lifetime of a resilient QEP when faced with unpredictable edge failures and predictable node failure events. .......................... 490

G.17 The experimental flow for determining the estimated lifetime of a resilient QEP when faced with unpredictable edge failures and predictable node failure events. .......................... 494
Abstract

Sensor networks comprise of a collection of resource-constrained, low cost, sometimes fragile wireless motes which have the capability to gather information about their surroundings through the use of sensors, and can be conceived as a distributed computing platform for applications ranging from event detection to environmental monitoring. A Sensor Network Query Processor (SNQP) is a means of collecting data from sensor networks where the requirements are defined using a declarative query language with a set of Quality of Service (QoS) expectations.

As sensor networks are often deployed in hostile environments, there is a high possibility that the motes could break or that the communication links between the motes become unreliable. SNQP Query Execution Plans (QEPs) are often optimised for a specific network deployment and are designed to be as energy efficient as possible whilst ensuring the QEPs meet the QoS expectations, yet little has been done for handling the situation where the deployment itself has changed since the optimisation in such a way as to make the original QEP no longer efficient, or unable to operate. In this respect, the previous work on SNQPs has not aimed at being resilient to failures in the assumptions used at compilation/optimisation time which result in a QEP terminating earlier than expected.

This dissertation presents a collection of approaches that embed resilience into a SNQP generated QEPs in such a way that a QEP operates for longer whilst
still meeting the QoS expectations demanded of it, thereby resulting in a more reliable platform that can be applicable to a broader range of applications. The research contributions reported here include (a) a strategy designed to adapt to predictable node failures due to energy depletion; (b) a collection of strategies designed to adapt to unpredictable node failures; (c) a strategy designed to handle unreliable communication channels; and (d) an empirical evaluation to show the benefits of a resilient SNQP in relation to a representative non-resilient SNQP.
Publications

Publications resulting from the work described in this thesis and the corresponding chapters in that these papers are built from are listed below:

- The contribution described in chapter 3 was published in the British National Conference of Databases (BNCOD) 2013 [109].

- The contribution described in chapter 4 has been submitted to the Scientific and Statistical Database Management Conference 2014 [111].

- The *unexpected node failure event* aspect of the contribution described in chapter 5 was published in the 10th International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE) 2011 [110].
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication
and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
Acknowledgement

I would like to express my deepest thanks to my supervisors, Dr Alvaro A. A. Fernandes and Prof. Norman W. Paton, whose guidance, friendship, patience and encouragement proved invaluable.

Thank you to my fellow colleagues and friends Alasdair Gray, Ixent Galpin, Christian Y. A. Brenninkmeijer, and Farahana Japeen who advised me throughout this project and whose friendship and opinions kept me on the straight and narrow during the project even when rubber monsters were in the air.

Thanks must go to all the people in The Information Management Group at the School of Computer Science of the University of Manchester to whom there are way too many to list individually but who’s support kept me in the project.

Thank you to my friends Klitos Christodoulou, Fernando Osorno, Khalid Belhajjame, Ruhaila Maskat, Rob Percival and many many more whose support, and rubber monster throws slowed me from falling off the road years ago.

I am extremely grateful for the funding supplied by the Engineering and Physical Sciences Research Council (EPSRC).

Thank you, my dear Olivia, for your love, affection, support and everything else in between; I know I would not have been able to make it this far without you.

And finally thanks must go to Emperor Shen Nung who discovered Tea in 2737 BC. Without his discovery, my world would have crashed down around me years ago.
Chapter 1

Introduction

1.1 Setting the Scene

Wireless sensor networks (WSNs) have become widely used in a range of applications, including data collection [81], event detection [103] and entity tracking [114]. They allow us to monitor the world discreetly and accurately, including areas that are hostile to or out of normal reach for humans. A WSN consists of a collection of static motes (see Fig 1.1) that can measure properties of the physical world through the use of on-board sensors [82]. Readings from these sensors are forwarded from the source motes through multi-hop communication, until they reach a special mote that handles the communication to a device outside the network (referred to as the sink/gateway node/mote or the base station).

Motes come with different capabilities, and typically have rather limited hardware resources, often consisting of a low-powered processor, a wireless transmitter and few kilobytes of programmable memory (e.g., a TelosB mote normally contains 48Kb [24]). They are usually powered by an attached battery pack, which, for a TelosB mote, contains two AA batteries. Other types of motes could be powered by vehicle batteries, solar panels, etc.
1.2 Motivation

To make use of a WSN, the end user must first purchase a collection of motes and batteries, then deploy them in the intended geographical area. This requires financial expenditure and, therefore, it is realistic to assume that the end user wishes to get the largest amount of data for their investment. Throughout the thesis, the amount of data to investment expectation is referred to as the “bang-for-your-buck” (BFB) metric. The expectation, or course, is that the more data is returned, the more benefit the user reaps, i.e., the greater the return on the investment made in acquiring and deployment the WSN. There is an implicit expectation too that this return will be greater the longer the lifetime of the deployment. In other words, this requires the deployment to be resilient to failure if the expected return on investment is to be achieved.

Unfortunately, the WSN infrastructure is fragile due to depletable energy stocks and the low-cost components used in motes. Therefore, motes often break down or malfunction. Additionally, as the communication between motes is wireless, it is susceptible to environmental noise and other types of interference.

1Note that papers often do not report failures in the WSN infrastructure directly, and, instead, characterise such failures as problems meriting future work.
which negatively affect the outcome (e.g., lead to loss of data packets).

Assume that a given WSN deployment costs $x$ to deploy for running an embedded program $q$ to produce data at a rate of $y$ tuples per second. Now, assume that at some time point $t$ a failure event $e$ causes $q$ to stop producing tuples (e.g., an essential relay node fails), then, this would result in a BFB value of $x / (y \times (t-t_0))$ where $t_0$ is the start time of $q$). By integrating techniques that adapt $q$ at (or immediately before) $t$ to a functionally-equivalent program $q'$ such that $q'$ compensated for the effects of $e$ and carried on producing tuples at the specified rate up to a point $t' > t$ the additional BFB value would be $x / (y \times (t' - t))$.

In short, adaptive techniques that increase the level of resilience to failure of a WSN deployment would, in general, improve its BFB value by increasing its lifetime, i.e., the period during which it produces the desired measurements. Such techniques would make WSN deployments more cost effective when confronted with failure events and, in turn, would make WSN deployments more viable in scenarios where such events are likely.

### 1.3 Sensor Network Query Processors

In the past, developing WSN applications was only possible with access to developers who had expert knowledge of the hardware interfaces, the underlying infrastructure and of application development in a distributed computing context. These applications would often take months to develop, due to cumbersome developer environments and would result in thousands of lines of embedded software code in low-level languages (e.g., NesC [41]), even for a simple data gathering application [21]. Once an application was written, it would remain unclear for how long the deployment might be expected to remain functional. This motivated the desire for some form of an abstraction layer that hid these hardware and software
complexities from an application developer, whilst ensuring that the end product met some functional and non-functional expectations. This was expected to facilitate the take-up of WSN technology by a larger population of non-technical users.

Sensor network query processors [32, 80, 77, 66] (SNQPs) are one such abstraction. SNQPs generate embedded software in the form of QEPs but hide the hardware complexities from the end user by allowing the functional requirements to be expressed in a declarative query language.

Because the motes that constitute a WSN deployment are often powered by batteries, a depletable energy source that often cannot be easily and cheaply recharged, energy efficiency has been the main focus of the WSN community as a performance metric [93]. Wireless communication is typically more energy-expensive than processing [92]. SNQPs save energy through their ability to reduce transmissions within the deployment, so that data is transmitted to the end user only if this is necessary to meet the declared requirements [47].

There are several SNQPs described in the literature (e.g., TinyDB [80], AnduIN [66] and SNEE [32]), that differ, e.g., in terms of what functionality to encompass in the QEP; what optimisations, if any, are applied to generate QEP; and how stringent the QEPs are in taking into account quality-of-service (QoS) expectations expressed by the end user.

QoS expectations are non-functional requirements placed upon a QEP [31]. They range from how often data is acquired from the sensors, to the maximum amount of time allowed for sensed data to be reflected in the results that reach the base station. A more complete discussion of existing types of QoS expectations in the literature is given in Section 2.2.2.

One of the main differences between SNQPs is the amount of knowledge of the current state of the WSN that they presume to have in order to generate optimised
QEPs. Some SNQPs (e.g., TinyDB, Smart CIS and AnduIN [80, 77, 66]) do not collect any data about the deployment before generating a QEP. This means that some optimisation is done at runtime thereby implying some overhead. Detection and adaptation to unexpected events that can affect the overall function of a QEP are likely faster in these architectures because the adaptations can be executed in a local manner (e.g., using clustering algorithms [93, 50]) in comparison to architectures where global knowledge is required before adaptations can take place.

The QEPs generated by SNQPs that do not gather data about the deployment before compilation may execute optimisations in a distributed manner at runtime, i.e. decisions are made locally. Examples of such decisions include what network operators to run on any given node, or how to avoid collisions through the use of media access control (MAC) protocols whilst, at the same time, ensuring that the QEP meets the expected delivery time [99]. Such QEPs are more prone to failing to meet the QoS expectations, or do not express any QoS expectations.

Other SNQPs, such as SNEE [32], try to avoid the shortcomings of local decision-making by compiling/optimising QEPs to a given deployment state. This requires the collection of metadata about the deployment beforehand for use at compile time. This metadata can include:

- The nodes that are operational in the deployment
- The energy reserves in each node.
- The sensing capabilities in each node.
- The available memory in each node.
- The communication edges that are operational in the deployment
1.3. SENSOR NETWORK QUERY PROCESSORS

<table>
<thead>
<tr>
<th>SNQP</th>
<th>Collect metadata</th>
<th>Level of in-network processing</th>
<th>Level of node failure support</th>
<th>Level of communication failure support</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyDB</td>
<td>No</td>
<td>Supports in-network queries, but with limited support for joins.</td>
<td>None</td>
<td>Allow nodes to select different parents</td>
</tr>
<tr>
<td>AnduIN</td>
<td>No</td>
<td>Supports some in-network operations that are executable in source nodes; other operations are carried out on the host machine.</td>
<td>Based on TCP/IP</td>
<td>Based on TCP/IP</td>
</tr>
<tr>
<td>SNEE</td>
<td>Yes</td>
<td>Supports very expressive in-network queries.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Smart CIS</td>
<td>No</td>
<td>Comparable to TinyDB</td>
<td>Executes a new route search when confronted with a parent node failure</td>
<td>Comparable to TinyDB</td>
</tr>
</tbody>
</table>

Table 1.1: Characteristics of existing SNQPs.

By compiling/optimising a QEP to a specific deployment state, the QEP can reduce to a minimum the need for runtime optimisation. For example, transmission events can be scheduled in the form of a global agenda, thereby minimizing the possibility of collisions at runtime. This allows the QEP, in principle, to more predictably meet QoS expectations. Therefore, this kind of SNQP is a more convenient environment to explore adaptive techniques for the purposes of resilience in real world deployments than SNQPs who optimise their QEPs at runtime.

Approaches based on collecting metadata beforehand have flaws too, of course, because now the QEP is optimised for a given deployment state. If the state of the deployment does change, such changes can result in the QEP becoming suboptimal, or even failing entirely. Because the physical infrastructure of a WSN is by intention, low cost, and hence of low quality, deployments are prone to change both in terms of what nodes are operational and in terms of what connectivity edges are operational and reliable. A summary of the behaviours of different SNQPs described in the literature, particularly with respect to how they handle QoS expectations and failure events in nodes or in communication edges is presented in Table 1.1.
Up to the point of writing this thesis, the state-of-the-art SNQPs [80, 32, 66, 77] exhibit very little resilience to predictable and unpredictable failures in the infrastructure in which they operate. This is due to the SNQP research mainly focusing on optimising the energy expenditure of their QEPs in such a way as to return the largest amount of data (defined as when the first participating node fails from energy depletion) within an ideal environment. Applications where a WSN is only considered as a data source from which data can be extracted, e.g., [82], also require techniques to adapt to environmental changes, and yet, these techniques are often unsuitable for direct application of SNQPs because they are intent on pushing computation into the WSN with a view to improving energy efficiency.

It is worth noting that it is often very difficult to compare the different SNQPs described in the literature because there is minimal intersection regarding the queries which are supported by all, or even many, of them. For such queries as are supported by a significant number of SNQPs, the internal behaviours are different enough, e.g., regarding optimisations at runtime (TinyDB, smartCIS and AnduIN), as to make performance comparisons difficult. An example of this is the comparison of tuple output for a query that is run on both TinyDB and SNEE. The difference in performance could be due to variations in the approach to either runtime optimisations in TinyDB (such as load shedding, adjustment of acquisition rate/delivery time, or routing decisions) against decisions made at compile time in the generation of the SNEE QEP. An attempt to compare different behaviours of in-network processing techniques was made by [35], which proved the point just made as to the challenges of carrying out such comparative evaluations.

There are many different types of events that can be experienced by a WSN which require actions to migrate them, such as hardware/software faults, loss of
sensor calibration, energy depletion, communication channel and arbitrary/unpredictable node failure. Some of these events are intermittent in the lifetime of the deployment e.g., a hardware fault may correct itself, or interference that was affecting the communication channel could have dispersed at a later time period. In this thesis the focus is on three of these failure events:

1. Node failure due to energy depletion.

2. Node failure due to arbitrary, permanent, and unpredictable node failure.

3. Unreliable communication channel failure.

These three events can be broken down into two distinctly different types of failure. Firstly, failures that can be predicted accurately using models, which in this thesis are referred to as \textit{predictable failure events}. Secondly, failures that cannot be predicted accurately using models, which in this thesis are referred to as \textit{unpredictable failure events}. Each of these failure events can be broken down into two subgroups, namely, node and communication channel failures. It is worth noting that at the time of writing, there was no way of accurately predicting communication failure events, and therefore that area is not explored in this thesis. This results in three research areas that are covered in the contribution chapters of this thesis (Chapters 3 to 5).

Work by the WSN research community can be categorised into two areas. The first includes proposals [13, 128, 37, 19, 63] that adapt with a view to reducing the energy or memory cost of the operations in a QEP [13, 128] or else speeding up their execution time [37]. They are, therefore, only indirectly intent on avoiding predictable node failure through the use of energy saving techniques. The second area includes proposals [14, 20, 56, 67, 123] that aim to deliver resilience for both
predictable and/or unpredictable failures. These proposals are designed to adapt to specific failure events, either of nodes or communication edges or both.

In the context of node failure, proposed techniques can be categorised into three distinct behaviours: firstly, techniques that adjust parts of a plan [14, 118]; secondly, techniques that adjust the routing between nodes [20, 25, 56, 3]; and finally, techniques that integrate the two types of behaviour [117]. These latter techniques are referred to as hybrid ones.

For communication channel failure, defining predictable and unpredictable is more difficult because modelling communication behaviour accurately is infeasibly hard without actually deploying the WSN in the real world and taking noise measurements before engaging in model construction. Therefore, predictable edge failures are construed to be those that are due to the physical properties of the link. Techniques for this are mainly proactive insofar as they either try to avoid using these edges (communication channels) altogether (by black-listing them at compile time) [67, 12], or else to compensate for the poor quality of the link by adjusting how much power is put into transmitting data through that edge [129]. In the case of unpredictable failures, all techniques must behave reactively and deal with the problem at runtime by adopting a best-effort approach [28], by multi-casting packets in the hope that some arrive at the base station [22, 123], or by retransmitting in an attempt to ensure that the data eventually arrives at the destination [108].

In this thesis, the detailed discussion of related work for each type of failure is given in the chapter that presents the corresponding contribution. Figure 1.2 presents a classification of techniques based on the types of failures they address. Later discussion whether or not such techniques can be applied in SNQPs and the advantages and disadvantages that might be expected from their application.
1.4 Aim, Objectives, and Contributions

The aim of the research presented in this thesis is to design and evaluate techniques for sensor network query evaluation that exhibit resilience to unpredictable and predictable failure events. Achieving this aim results in an increase in the functional lifetime of a WSN deployment beyond what is possible by the current state-of-the-art SNQPs, thereby making SNQP technology more cost-effective.

The research aim was decomposed into the following objectives, from which corresponding research contributions were obtained that the remainder of the thesis describes and evaluates:

**O1** To design and evaluate techniques for adapting to unpredictable node failure events to extend the functional lifetime of WSN deployments whilst ensuring that the QEPs meet QoS expectations.

The research contributions stemming from this objective are as follows:

1. A technique that, in response to a node failure event, re-compiles and re-optimises the query in order to obtain a new QEP for the new state of the deployment, and ships the new QEP to the nodes that need reprogramming,
resulting in the resumption of the evaluation of the query thereby extending
the functional lifetime of the deployment with respect to a non-adaptive
approach. This contribution is discussed in Chapter 3.

2. A technique that generates a new, functionally-equivalent QEP that is as
similar to the previously-running QEP as possible, but adapted to the new
state of the deployment. The technique has the implicit aim of reducing
the need for runtime reprogramming. Since the new QEP resumes the
evaluation of the query, the functional lifetime of the deployment is thereby
extended with respect to a non-adaptive approach. This contribution is
also discussed in Chapter 3.

3. A technique that exploits deliberate, planned node redundancy in WSN
deployments with a view on improving resilience to node failure. The tech-
nique requires no run-time reprogramming and ensures a minimum level $k$
of resilience to failures, enabling at least $k$ QEP adaptations, each of which
is potentially capable of extending the functional lifetime of the deployment.
This contribution is discussed in Chapter 5.

O2 To design and evaluate techniques for mitigating predictable failure events
in the context of nodes in SNQP QEPs with the aim of extending the
functional lifetime of a WSN deployment whilst ensuring that the QEPs
meet QoS expectations.

The research contribution stemming from this objective is a technique that,
at compile time, plans a sequence of adaptations between different, yet function-
ally equivalent QEPs, so as to delay the occurrence of node failure events as a
consequence of energy depletion. The technique relies on predictive energy con-
sumption models, and results in an extension of the functional lifetime of WSN
deployments. This contribution is discussed in Chapter 4.

O3 To design and evaluate a technique for adapting to (partial or complete), unpredictable failure events in the context of communication channels (referred to as link failure from now on).

The research contribution stemming from this objective is a technique that uses deliberately planned node redundancy in WSN deployments with a view of exploiting the additional communication paths as potentially useful for retransmissions, thereby increasing the overall numbers of tuples delivered during the functional lifetime of the deployment. This contribution is also discussed in Chapter 5.

1.5 Overview of Thesis Structure

The background to the research is discussed in Chapter 2. Section 2.1 introduces different forms of query processing, viz., classical, distributed, and stream-based. Section 2.2 gives a detailed description of how to construe a sensor network in the context of this thesis and the types of applications for which they are commonly used. In Section 2.2, the concept of QoS in WSNs is described in detail and, finally, in Section 2.3 a description and discussion on why a specific SNQP was chosen as the basis for the experimental work in this thesis is presented. The chapter presents some conclusions in Section 2.4.

For each of the contribution chapters (Chapters 3 to 5) the structure is fundamentally the same, in that first there is a discussion on the types of approach that may be suitable for a SNQP context for the type of event being processed. A description of how such an approach can be integrated into a SNQP processing stack (in this thesis the SNQP in question is SNEE) follows. A description of
the techniques involved in the chapter is presented and then evaluated. After
the evaluation, related work is discussed and conclusions made. Each chapter
contains its own related work, and evaluation, as each problem is relatively self
contained.

Overall conclusions are presented in Chapter 6. This chapter reflects on the
implications of the results with respect to the design of SNQPs and to the design of
the WSN deployment on which QEPs execute. The main conclusion is that with
techniques contributed in this thesis, the functional lifetime of the deployment
can be significantly increased with respect to what can be achieved by the state-
of-the-art SNQPs. Using these techniques the overall BFB, in the presence of the
real world challenges of certain failure events, can be significantly increased. By
using resilient SNQPs the scope of applications to which SNQP technology can be
applied in a cost-effective manner is significantly increased. Finally, future work
is considered that might follow from the research contributions in this thesis.
Chapter 2

Background

This chapter covers the background knowledge that underpins the research contributions reported in the remainder of this thesis. Note that this chapter does not describe potentially competing research; this is done inside the contribution chapters. Section 2.1 summaries the different types of query processing that give rise to SNQPs. Section 2.2 describes WSNs, including the types of deployments currently envisaged, and the QoS expectations typically placed on these deployments. Section 2.3 describes the SNEE SNQP compilation stack, into which all of the contributions reported in this thesis have been integrated. Finally, Section 2.4 reviews how the database research community has applied the techniques described in Section 2.1 to WSNs by construing the latter as a distributed computing platform, over which queries are posed and executed. Current SNQPs have not focused on counteracting the unreliability of the physical fabric of WSNs. The contributions reported in this thesis advance the state-of-the-art in this particular respect as Chapters 3 to 5 demonstrate.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Notation Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>scan $\sigma_C(R)$</td>
<td>Read tuples from relation $R$, discarding those that do not meet conditions in predicate expression $C$.</td>
</tr>
<tr>
<td>select</td>
<td>$\sigma_C(R)$</td>
<td>Discard tuples in relation $R$ which do not meet conditions in predicate expression $C$.</td>
</tr>
<tr>
<td>project</td>
<td>$\pi_L(R)$</td>
<td>Remove any attributes in $R$ which are not in the attribute list $L$.</td>
</tr>
<tr>
<td>aggregate</td>
<td>$\gamma_{F,L}(R)$</td>
<td>Aggregate similar values from a collection of tuples into a single value using the merge function $F$ (e.g., SUM, COUNT or AVG). If a list $L$ of grouping attributes is specified, a result is given for each distinct value of the grouping attribute(s).</td>
</tr>
<tr>
<td>natural join</td>
<td>$R \Join S$</td>
<td>Concatenates tuples from $R$ and $S$ which have equal values for attributes with the same name.</td>
</tr>
<tr>
<td>theta join</td>
<td>$R \Join_\theta S$</td>
<td>Concatenates tuples from $R$ and $S$ for which condition $\theta$ is true. $\theta$ is a condition over attributes in relations $R$ and $S$, e.g., $R.x &gt; S.y$.</td>
</tr>
<tr>
<td>sort</td>
<td>$\text{sort}(R)$</td>
<td>Returns the tuples of $R$ in sorted order.</td>
</tr>
<tr>
<td>distinct</td>
<td>$\delta(R)$</td>
<td>Returns the tuples in $R$ with duplicates removed.</td>
</tr>
</tbody>
</table>

Table 2.1: Logical query operators.

2.1 Query Processing

This section describes the different types of query processing that converge and give rise to SNQPs. It comprises of three parts: Section 2.1.1 describes classical query processing, where queries are evaluated against a single, centralised relational data store. Section 2.1.2 describes distributed query processing, where queries are evaluated against data stores that are located on distinct, separate, autonomous sites. Section 2.1.3 describes query processing over streams of data, where the data continuously arrives for processing and cannot, therefore, be held in finite data stores.

2.1.1 Classical Query Processing

The material in this subsection is summarized from [40]. The compilation process that determines how a query is to be executed can be broken into two stages. Classical query processing consists of querying a centralised data store where
2.1. QUERY PROCESSING

queries are formulated in a declarative language that describes what should be returned, such as SQL [49].

1. The first stage translates the query into a logical form, in which the order of the logical operators that expresses $q$ (of which, classical examples are illustrated in Table 2.1) has been optimised through the use of heuristics and equivalence relations to reduce the overall workload, thereby fostering reduced query response times. An example of a logical query plan (LQP) for the query represented in Figure 2.1(c) is given in Figure 2.1(a).

2. The second stage takes the LQP and translates it into a physical query plan (PQP), where a physical algorithm is selected as the chosen implementation for one or more logical operators. The selection process uses performance statistics for each algorithm in past workloads, e.g., the memory required and the speed of execution for each algorithm. These statistics are then used as inputs to an analytical estimation model that assigns the predicted response time as the cost of the PQP. It is worth noting that the mapping from a LQP to PQP is a one-to-many relationship and, therefore, a collection of PQPs are assigned a cost, so that the cheapest can be selected for actual execution. An example of a PQP generated from the LQP in Figure 2.1(a) is in Figure 2.1(b).

Once a PQP has been selected, it is executed over the data store, and results are returned to the end user.

2.1.2 Distributed Query Processing

Due to the great success of classical query processing, classical database technology is used in many areas of industry, and it was therefore only a matter of
(a) Logical Form

(b) Physical Form

(c) Example SQL Query

```
SELECT f.id, f.light, c.id, c.light,
FROM field f, forest c
WHERE f.light > c.light
```

Figure 2.1: An example query at different stages of classical query processing.
time until users wished to query data over several data stores that were located on different physical machines. This has now become common in certain areas of industry (e.g., financial and scientific), and has been the motivation behind distributed query processing (DQP). This required modifications to the classical query processing approach as the QEPs now need to be compiled / optimised to account for the data potentially being spread over several different machines with different processing capabilities, spread over a wide geographical area.

A two-stage optimisation paradigm has been used in DQP very successfully, where the first stage executes the optimisation steps from the classical query processing approach discussed previously, resulting in a single site PQP; the second phase makes the decisions required to convert the single site PQP into a distributed query plan (DP), where the operators are grouped into fragments, and the fragments scheduled for execution at the required sites, so that the overall cost of the QEP is kept as low as possible.

For example, in Figure 2.2, there is an underlying fabric involving six sites. There are three fragments, represented by the rounded boxes, referred to as F1, F2, and F3. Fragment F1 is located on sites 5 and 1 (as denoted by the numbers in curly brackets at the bottom right hand corner of the fragment box), fragment F3 is located on sites 7, 3 and 4, and fragment F2 is located on site 8. Fragment F1 and F3 are scheduled on the sites that contain the extents forest and field, to reduce communication costs. The NESTED_LOOP_JOIN that gives rise to Fragment F2 requires input from all five PROJECT operators to execute. It, therefore, has its own fragment, and possibly because sites 7, 3, 4, 5 and 1 might not have enough main memory to hold the structures used by the NESTED_LOOP_JOIN, it is scheduled to site 1 where the fragment runs alone.

1Recall that there are three copies of one left input and two copies of the right input
Figure 2.2: The example query in distributed form.

Note that data flow between sites is handled by EXCHANGE operators [48]. An EXCHANGE operator is broken down into two dependent processes, named a producer and a consumer. The producer is responsible for sending tuples, and the consumer for receiving them and storing them for the parent fragments to consume.
2.1.3 Stream Query Processing

Data generating technology has integrated itself into our daily lives. The speed at which and, therefore, the quantity of data that can be generated from a multitude of different sources are such that data can no longer always be stored to be queried later. Data processing has to be done, more and more, in near real time. Queries are also expected to execute over large amounts of real-time data and to return results in real time too. One example of this new class of applications is the processing of information stemming from continuous data feeds from financial and commodity markets. This has resulted in the need for query processing techniques that operate over streams of data with unknown, potentially unbounded cardinality (as opposed to persistent data stores and that require queries to be evaluated again and again with different parameters or in response to new data becoming available. Research prototypes such as Aurora [2], STREAM [6] and TelegraphCQ [18] were developed as data stream management systems (DSMSs). DSMSs differ in how they compile and execute queries in relation to previously mentioned types of query processing in that:

- The tuples have to be processed in real time, as the data is based on real world events to which end users wish to react in real time.

- Due to the impracticability of storing all the tuples, if queries are not executed on tuples when they arrive, the tuple may never be processed and be lost forever.

- Tuples are pushed onto the query processor instead of being pulled from data stores by the query processor, as a result of which it becomes more difficult to estimate and control primary memory occupancy.

- Tuples cannot form sets, rather they form time-stamp-ordered sequences
and, therefore, have an explicit ordering.

The potentially unbounded cardinality of the streams causes new challenges for DSMS, in that many operators defined in classical relational algebra exhibit blocking behaviour, i.e., at least one input must be entirely consumed before the first output tuple can be emitted. Sort and some implementations of join operators (e.g., nest loop join [40]) are examples of blocking operators. These operators are only meaningful if they are executed over defined sections $s'$ of the stream $s$, during which the tuples in $s'$ are stored in memory for the duration of the operator’s execution. If not all the tuples of $s$ can be part of $s'$ for the time available for the operator to evaluate, then any result will be approximate. To generate $s'$, DSMSs use the notion of a window, and typically a sliding window, insofar as its contents change as time progresses and more data arrives. Windows break a stream into sections, whose type is conveyed by the declarative language used to express the query. Using windows, a user can pose such queries as:

- Report, every five seconds, the average of all FTSE company share prices.
- Report, every hour, the temperature at the mouth of the volcano.
- Report, every hour, the distance travelled over the last day, by a given point of a moving glacier.

As data is constantly being pushed onto the query processor, queries must be evaluated in response, resulting in what is called a continuous query. Such a query revises the classical compile-once, execute-once paradigm into a compile-once, execute-many paradigm.

In most stream applications, sources are over a geographically large area, therefore DSMSs such as Borealis [1] use distributed processing techniques discussed previously to optimise their QEPs for the given architecture in terms of
balancing workload [113], or adapting to faults [9].

2.2 Wireless Sensor Networks

This section describes how a WSN is construed in the context of this thesis and how different types of WSNs have been used in different applications. Section 2.2.1 discusses how a WSN is constrained by the limited capabilities of its constituent parts, and Section 2.2.2 surveys the different types of QoS expectations that maybe placed upon different WSN applications.

2.2.1 Deployments

WSNs have been used in many applications, such as discreetly observing wildlife [82], locating snipers in a military context [103] and monitoring crops [81]. A large community of WSN users are scientists that use them for monitoring environmental phenomena. Recently, other applications have begun to emerge, ranging from monitoring availability of car park spaces [11] though to alerting users of obstacles in a mobile setting [72]. This thesis focuses on fixed, mote-level WSNs in which, throughout the lifetime of the deployment, there is a significant practical benefit stemming from self-repair and self-optimisation capabilities so that there is a better return on the investment made by the deployer.

A WSN, in the context of this thesis, consists of a collection of motes that work together to perform a given data collection task, which requires monitoring and processing sensor data gathered from the environment in which the motes are placed. Each mote contains a small amount of RAM memory (e.g., 10kb [24]), and of programmable flash memory (e.g., 48 kb [24]), a low-power, low-frequency wireless radio transceiver, a processor and a depletable energy source (e.g., two AA batteries). A mote may also contain various sensors to generate
readings about the environment they are immersed in (e.g., a pressure, light or temperature). Overall, motes provide a very resource-constrained platform, so that any task that is allocated to them must be optimised to spend as little energy as possible, with a view to reducing the risk of energy depletion before the WSN task is completed.

In the context of this thesis, a WSN is by default heterogeneous, in the sense that motes can have:

- different, or no, sensors on board,
- different amounts of available memory, and
- different amounts of available energy.

Motes that do not contain any sensors on board can be used as either computation sites or as relays. In the former case, they are allocated computation tasks on inputs they receive to produce outputs they transmit onwards. In the latter case, more simply, they simply receive and transmit data. In both cases, the use of non-sensing nodes increases the multi-hop options of a WSN. In each WSN, there is at least one mote that handles communication with the encompassing environment so as to report results or to request/receive commands from a controller equipment. These mote(s) are known as gateway motes, or base stations. In this thesis a single base station is assumed.

As the motes operate with a depletable energy source, there has been much research into making task execution as energy efficient as possible. At the level of hardware, motes use low-power processors and radios. At the level of software, one important goal is to duty-cycle, i.e., move a hardware component to a low-power state when it is not in use. In this respect, because in motes computation
is typically less energy-expensive than communication, it is desirable to use compu-
tation more intensively to reduce/filter the amount of data that requires to be communicated [92]. SNQPs have been proposed as particular appropriate to meeting this goal.

Motes often consist of a circuit board, with a single battery pack attached to it, as shown in Figure 1.1, and are rather fragile. In spite of this, there is a need to deploy them in remote, and often hostile environments (e.g., on volcanos or in forests), where battery replacement is often too costly and where motes are exposed to the elements and to wildlife hazards. This means that motes often fail due to their energy supplies being depleted, or from unforeseeable mishaps (e.g., being trampled upon or being otherwise destroyed by wildlife). This poses additional challenges to any WSN applications, as they must be designed to be resilient to predictable and unpredictable changes in the computational fabric in which they are executed. There has been some research into developing applications that adapt to a changing infrastructure, but that literature sees WSNs as simple data sources that make little use of computation inside the WSN. These techniques are often not suitable in the context of SNQP, due to their often high energy demands, and their lack of QoS awareness. These techniques are discussed in more detail in Sections 3.5, 4.5 and 5.6.

### 2.2.2 Quality of Service in Wireless Sensor Networks

As in any application deployment, in WSN applications too the end user wishes for the most gain for their financial outlay, viz., the largest amount of relevant sensor data for the cost of the deployment (which in this thesis is referred to as the BFB metric). This high return on investment is an example of a QoS expectation.
The requirements of a software package can be broken down into two types: functional and non-functional. Functional requirements define the specific purpose of the application, i.e., how that application should respond to a given set of inputs. The non-functional requirements define how the end user expects the application to perform while carrying out its functional requirements. In this sense, QoS expectations are a kind of non-functional requirement as the level at which an application meets them will affect the opinion of the end user as to whether the application is behaving appropriately or not. For example, assume an application with the QoS expectations that motes will sense the environment every ten seconds and the application will deliver results every hour. If, after a period of time, the motes started sensing every minute and delivering results every two hours, the end user would consider the application to have violated the QoS expectations placed upon it.

Romer and Mattern [96] present a survey in which they define several non-functional requirements for sensor networks, of which a selection is listed:

**Acquisition interval** i.e., the length of time between consecutive sensor readings. We denote it by $\alpha$.

**Delivery time** i.e., the maximum length of time from which a sensor reading must take to reach the end user. We denote it by $\delta$.

**QEP lifetime** i.e., the length of time a QEP can run until the first mote has failed. We denote it by $\lambda$.

**Deployment lifetime** i.e., the length of time the deployment can run a given query until enough motes have failed for the deployment to be unable to execute the query successfully.
2.2. WIRELESS SENSOR NETWORKS

**Coverage** i.e., the number of sensors actively covering a geographical area at any given point in time within the deployment lifetime.

**Completeness/ accuracy** i.e., how complete or accurate a given result delivered to the end user is.

A subsection of these will be in scope in later chapters of this thesis. In Table 2.2, a collection of applications and some QoS expectations that one might expect to be placed upon them are presented. The applications are:

- A deployment on the Great Duck Island that was used to monitor Leach’s storm petrel’s behaviour and habitat [82].
- A tracking deployment that fitted free-roaming zebras with GPS collars. Communication between the motes was intermittent as they only communicated with each other if and when they were in radio range [127].
- A military deployment that localised the position of a sniper in a village [103] through audio sensors.
- A deployment that monitored volcanic and seismic events in Ecuador [120].
- A deployment designed to monitor glacier movement [84] over long periods of time in Norway.

In all cases, the acquisition rate and delivery time vary, ranging from milliseconds to days. Conversely, the expected functional lifetime is fixed at a minimum level, therefore any improvement is desirable. While only the Great Duck Island deployment made use of first generation SNQP technology, it is instructive to consider how SNQPs would have fared in the other deployments above. Any QEP in any of the applications, apart from the ZebraNet application, would have been
<table>
<thead>
<tr>
<th>Application</th>
<th>Sensing Modalities</th>
<th>Acquisition Interval</th>
<th>Delivery time</th>
<th>Energy Stock</th>
<th>Lifetime</th>
<th>Main QoS of concern</th>
<th>Description of Sensing Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Great Duck Island [82]</td>
<td>Light, temperature, infrared, relative humidity, and barometric pressure</td>
<td>5 min to 1 hour, depending on query</td>
<td>Real-time</td>
<td>2 AA batteries (2200 mAh) per mote</td>
<td>9-12 months</td>
<td>Lifetime</td>
<td>100+ motes at fixed locations</td>
</tr>
<tr>
<td>Zebranet [127]</td>
<td>GPS</td>
<td>8 minutes</td>
<td>Best effort</td>
<td>2 mAh lithium ion battery per mote charged by solar panels</td>
<td>1 year</td>
<td>Lifetime</td>
<td>7 motes moved by zebras over large area</td>
</tr>
<tr>
<td>Glacier Monitoring [84]</td>
<td>Pressure, temperature, orientation (tilt in three dimensions), external conductivity and strain gauge</td>
<td>4 hours</td>
<td>1 day</td>
<td>Lithium Thionyl Chloride battery</td>
<td>1 year</td>
<td>Lifetime</td>
<td>9 motes moved by ice</td>
</tr>
<tr>
<td>Sniper Localization [103]</td>
<td>Acoustic</td>
<td>Continuous</td>
<td>Two seconds or less</td>
<td>Two AA batteries per mote</td>
<td>Weeks or months</td>
<td>Acquisition interval, delivery time, lifetime</td>
<td>60 motes over 80m x 80m area</td>
</tr>
<tr>
<td>Volcán Reventador [120]</td>
<td>Seismometer and microphone</td>
<td>10ms</td>
<td>Non-event data stored at source mote; events notified to gateway within 490s</td>
<td>Two D-cell batteries per mote</td>
<td>19 days (batteries replaced weekly)</td>
<td>Acquisition interval, delivery time</td>
<td>16 motes over 3Km section of mountain</td>
</tr>
</tbody>
</table>

Table 2.2: Characteristics of existing sensor network deployments as collated in [31].
expected to ensure that the acquisition rate and delivery time are met throughout
the lifetime of the application. This highlights the need for resilient behaviours
to be integrated into the operating QEPs, as failure to adapt to failure events
can cause the QEP to fail to meet QoS expectations even if, functionally, results
continue to be produced. For example, if a node were to fail, the QEP might fail
to meet the delivery time or the functional lifetime.

2.3 **SNEE SNQP**

This section describes the **SNEE SNQP** within which all of the contributions of
this thesis have been implemented.

**SNEE** has been developed at the University of Manchester for almost ten years.
It has been described in several papers [15, 33, 16, 36, 34, 32]. In this section, the
description of the **SNEE SNQP** is strictly confined to describing characteristics,
properties and functionalities as they existed prior to the research carried out
by the author, from which the contributions reported in Chapters 3 to 5 have
emerged. The level of technical detail used is higher than in the rest of this
chapter, but it is fundamental to allow the reader to fully comprehend the novel
contributions reported later in the thesis.

Section 2.3.2 discusses the inputs that the **SNEE SNQP** requires to optimise
its QEPs, to take into account the stream and distributed nature of WSNs. Sections 2.3.3 and 2.3.4 explain in detail the steps involved in the **SNEE** compilation/optimisation stack. This section is, therefore, an overview of the **SNEE**
literature [15, 33, 16, 36, 34, 32] adapted for the purposes of this thesis.
CHAPTER 2. BACKGROUND

2.3.1 SNEE overview

The SNEE SNQP aims to generate energy- and memory-efficient QEPs through the optimisation techniques that are integrated into its QEPs. An example of a memory-saving technique is that the compiler spreads the computation over several motes and only the minimal amount of code required to execute is placed in each mote. This results in lean memory footprints in each mote, which, in turn, allows data to be buffered in the mote’s memory. Buffering data and communicating it in bigger blocks reduces energy expenditure in communication and leads to significant savings so long as the delivery time QoS expectation is met.

An energy-saving technique in SNEE is the use of a fixed agenda that determines when each computation is executed and when data is transmitted between motes. By having fine-grained control over wireless communication, the SNEE compiler can avoid the situation where multiple motes transmit data at the same time and thereby interfere with each other (leading to collisions) or having to drop data due to being overloaded at runtime (requiring load shedding). Finally, through the use of analytical cost estimation models, the SNEE compiler can determine the minimal length of time required to have the radio on to transmit and receive data, thereby saving energy by not having mote components active any longer than they need to.

These energy- and memory-saving techniques allow the SNEE QEPs to guarantee, in an ideal environment, that data is delivered to the end user, according to the QoS expectations.

It is worth noting that the SNEE compiler/optimiser differs from some SNQPs (e.g., AnduIN [66], TinyDB [80] and SmartCIS [77]) in not using standard protocols to control the communication between motes or an interpreter in each mote to determine what operations to carry out. This means that direct comparison
of SNEE and its extensions as described in this thesis with other SNQPs is difficult to perform because of the distinctly different design choices and operational behaviours.

As mentioned previously, the SNEE compiler/optimiser, as described in [15, 33, 16, 36, 34, 32], does not generate adaptive QEPs. This is apparent in Figure 2.3, where it can be seen that once the binaries have been shipped to the WSN deployment (denoted by Stage 1.2 in Figure 2.3), the only interaction that the SNEE runtime has with the QEP is for the purpose of receiving tuples. Throughout the thesis, the infrastructure diagram in Figure 2.3 is updated to show what changes are required for each strategy to operate successfully.

Figure 2.3: The SNEE Infrastructure.

### 2.3.2 SNEE Inputs

The compilation/optimization process takes a collection of inputs that allow the compiler to optimise its QEPs for the given deployment. The inputs are as follows:
CHAPTER 2. BACKGROUND

1. A SNEEq query (as exemplified in Fig. 2.4(b)), QoS expectations in the form of a desired acquisition rate (i.e., the frequency at which sensing takes place) and a maximum delivery time (i.e., an upper bound on the acceptable amount of time between data being acquired and being reflected in the emitted results).

2. The current connectivity graph, which describes what motes exist in the deployment as well as which motes can communicate with each other through the use of the cost-labelled edges, as shown in Figure 2.4(a).

3. The logical schema for the query, which describes the available logical extents over the sensing modalities in the deployment, as shown in Figure 2.4(b).

4. The physical schema for the query, which describes which physical nodes contribute readings to which logical extent, and which node acts as the base station, as shown in Figure 2.4(b).

5. Statistics about each node (e.g., available memory and available energy stocks).

6. Cost-model parameters used in the compilation process (e.g., unit energy costs for sleeping, sensing, processing, and communicating) [16].

The example query in Figure 2.1(c) takes two data streams, one stemming from sensors in a field, the other from sensors in a forest, with both kinds sensing the environment every 5 minutes. The query selects tuples for which light levels are higher in the forest than in the open field at the same time point and emits them onto the output stream. The intuition behind the query is that if light levels in the forest are higher than in the open field, then one might suspect that a forest
2.3. SNEE SNQP

fire has started. A breakdown of the query operators supported by the sneeql language are represented in Figure 2.3, a more through description of the sneeql language can be found in appendix F.

<table>
<thead>
<tr>
<th>Stream-to-Stream Operators</th>
<th>Stream-to-Window Operators</th>
<th>Window-to-Stream Operators</th>
<th>Window-to-Window Operators</th>
<th>Any-to-Same-as-Input-Type Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a reading every AcqInt from sensors in AttrList and apply SELECT[PredExpr] and PROJECT[ProjList] in that order on the resulting tuple.</td>
<td>Define a time-based window on S from startTime to endTime inclusive and re-evaluate every slide time units.</td>
<td>Emit onto S all the tuples in W.</td>
<td>Using the nested-loop join algorithm, emit onto the output the concatenation of each tuple from the left to each tuple from the right input (keeping only the attributes in ProjList) if it satisfies PredExpr.</td>
<td>Emit onto the output every tuple from the input that satisfies PredExpr.</td>
</tr>
<tr>
<td><strong>DELIVER</strong>[(S) : S]</td>
<td><strong>ROW_WINDOW</strong>(startTime, endTime, slide)(S) : W</td>
<td><strong>ISTREAM</strong>[(W) : S]</td>
<td><strong>AGGR_INIT</strong>(AggrFunction, ProjList)(W) : W</td>
<td>Emit onto the output a tuple formed with the attributes from the input tuple that occur in ProjList.</td>
</tr>
<tr>
<td>Deliver the query results.</td>
<td>Define a tuple-based window on S from startTime to endTime inclusive and re-evaluate every slide time units.</td>
<td>Emit onto S the newly-inserted tuples in W since the previous window evaluation.</td>
<td>Merge into the partial result the values from input for attributes in ProjList for type of aggregation specified by AggrFunction.</td>
<td>Emit onto the output the final result of incrementally aggregating the attributes in ProjList for type of aggregation specified by AggrFunction.</td>
</tr>
<tr>
<td><strong>TRANSMIT</strong>[(X) : X]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pack input tuples into blocks up to the maximum packet size and send them over the radio.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RECEIVE</strong>[(X) : X]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive blocks of up to the maximum packet size, unpack the tuples and emit them.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RELAY</strong>[(X) : X]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive blocks of up to the maximum packet size, then sends them over the radio.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: SNEEql Physical Algebra.
CHAPTER 2. BACKGROUND

(a) Example Connectivity Graph

Logical Schema:
field (id, time, temp, light);
forest (id, time, temp, light);
Physical Schema:
field: (N5, N1);
forest: (N4, N3, N7);
sink: (N0)

Q: SELECT RSTREAM fi.id, fi.light,
fo.id, fo.light
FROM field[NOW] fi, forest[NOW] fo
WHERE fo.light > fi.light
AND fi.light > 5

QoS Expectations
(Acquisition Rate = 5 Minutes,
Delivery Time = 1 hours)

(b) Example SNEEq Query, QoS Expectations, Logical and Physical Schemas

Figure 2.4: Examples of SNEE inputs.

The SNEE compilation stack, depicted in Figure 2.5, follows the same approach as the DQP techniques discussed in Section 2.1.2, in that it has two phases: a single-site phase which generates a physical algebraic form of the query (known as PAF, comparable to the PQP described in Section 2.1.1) followed by a multi-site phase that converts the PAF into a distributed algebraic form (known as DAF, comparable to the DP described in Section 2.1.2), and makes decisions that are specific to operating on a WSN such as:

- How to route tuples.
- Where to place fragments.
- When to transmit them.
Figure 2.5: The SNEE compilation/optimisation stack [110].
2.3.3 **SNEE single site optimisation phase**

The single-site phase, described in Stages 1 to 3 in Figure 2.5, performs the classical query optimisation processes of:

1. Validating the **SNEEql** query syntax and generating the corresponding abstract syntax tree.

2. Translating the abstract syntax tree into a logical algebraic form (LAF) and rewriting the LAF to reduce intermediate results. The resulting operator tree is referred to as the logical algebraic form (known as LAF, comparable to the LQP described in Section 2.1.1) of the query, as illustrated in Figure 2.6(a).

3. The selection of physical algorithms used to implement each logical operator resulting in the PAF of the query, as illustrated in Figure 2.6(b).

As the query is compiled for a WSN, rather than scan operators, ACQUIRE operators are used, which reflect the fact that tuples are being periodically generated, at the stipulated acquisition rate, from a sensor. An ACQUIRE can also encapsulate SELECT and PROJECT operators. Due to the data being streamed, SNEE uses WINDOW operators to partition a stream into windows that are processable by the traditional relational operators (e.g., joins) and uses CQL-inspired [7] operators to convert windows back to streams. Finally, a DELIVER operator serves tuples to the base station. A more in-depth explanation of the **SNEEql** operators and algebra can be found in Appendix F.

2.3.4 **SNEE Multi-Site Optimisation Phase**

The multi-site phase, depicted by Stages 4 to 7 in Figure 2.5, converts the PAF into the distributed algebraic form (DAF) of the query, which takes into account
the challenges faced when operating in a WSN deployment. These include how to transmit the data from the distributed sources up to the base station in multi-hop fashion whilst processing tuples through in-network operators in an energy-efficient manner and ensuring tuple transmissions do not interfere with each other.

Each of these decisions is now discussed in detail.

2.3.4.1 Routing

The routing stage of the SNEE compilation/optimisation stack generates a routing tree (RT) for the query that is an approximation of a Steiner tree [65]. A Steiner tree is a minimum spanning tree (and hence is likely to be energy-efficient) that includes a given set of essential nodes, known as Steiner nodes.

The routing algorithm (shown in Figure 2.8) makes the base station, which is one of the Steiner nodes, the root of the RT, and then ensures that the remaining Steiner nodes, one by one, are part of the RT by finding the shortest path between that and some node already in the RT, adding to the RT all the nodes in that
For the example query in Figure 2.4(b), Figure 2.7 presents the generated RT over the connectivity graph in Figure 2.4(a). In the RT, each vertex is a WSN node; a direction edge denotes that the child sends data to the parent; double-line circles denote acquisition nodes that participate in the extents required by \( q \) and the base station; single-line circles denote nodes that relay tuples from the acquisition nodes to the base station. The Steiner nodes in this example are nodes 6, 9 and 7, i.e., the acquisition nodes for the extents FOREST and FIELD, and node 8, i.e., the base station. The shortest path between nodes 7, 6, 9 and 8 is through nodes 3 and 1 in this case. Nodes in the connectivity graph that are not selected to participate in the RT remain idle, an energy-saving state, for possible use in a later query.
2.3. SNEE SNQP

Routing($P_Q, G$)
1. $\triangleright$ Compute the approximate Steiner tree ($rtV$, $rtE$)
2. $\triangleright$ for ($G$,$P_Q$.Sources $\cup$ {$P_Q$.Destination} ).
3. $rtV \leftarrow \{P_Q$.Destination$\}$
4. $rtE \leftarrow \emptyset$
5. remaining$V \leftarrow P_Q$.Sources
6. while remaining$V \neq \emptyset$
7. $\triangleright$ from $\leftarrow$ ChooseOne(remaining$V$)
8. $\triangleright$ to $\leftarrow$ ChooseOne(rt$V$)
9. $\triangleright$ path $\leftarrow$ Shortest-Path(from, to, $G$)
10. $rtE \leftarrow rtE \cup$ EdgesIn(path)
11. $\triangleright$ rt$V \leftarrow rtV \cup$ VerticesIn(rt$E$)
12. remaining$V \leftarrow$ remaining$V \setminus rtV$
13. return (rt$V$,rt$E$)

Figure 2.8: An Algorithm for Computing a Routing Tree.

2.3.4.2 Where-Scheduling

The where-scheduling stage of the SNEE compilation/optimisation stack is broken down into two separate stages: partitioning and fragment allocation.

Partitioning defines the fragmented algebraic form (FAF) of a PAF by breaking up selected edges ($child$, $op$) $\in$ PAF into a path $[(child,e_p),(e_c,op)]$ where $e_p$ and $e_c$ denote, respectively, the producer and consumer parts of an EXCHANGE operator. Each operator is considered to be of one of four types. These are:

**Location-sensitive operators** which are operators where there is no leeway as to which WSN nodes the operator can execute on. Examples of location-sensitive operators are the ACQUIRE and DELIVER operators.

**Attribute-sensitive operators** which are operators where the partitioning of tuples is important to the semantics of the operator when there are multiple copies of the same operator [54]. An example of an Attribute-sensitive operator is the JOIN operator.
\textbf{Fragment-Definition}(\textit{P}, \textit{Q}, \textit{Size})

1. \[
F_Q \leftarrow P_Q
\]

2. \textbf{while} \begin{itemize}
\item \textgreater{} post-order traversing \(F_Q\),
\item \textgreater{} let \(op\) denote the current operator
\end{itemize}

3. \textbf{do for each} \textit{child} \in op.	extit{Children} \\
4. \textbf{do if} \textit{Size}(\textit{op}) \geq \textit{Size}(\textit{op}.	extit{Children}) \textbf{or} \textit{op}.	extit{LocationSensitive} = \textbf{yes} \\
5. \hspace{1em} \textbf{or} \textit{op}.	extit{AttributeSensitive} = \textbf{yes} \\
6. \hspace{2em} \textbf{then} \textbf{Delete}(\textit{child}, \textit{op}, \textit{P}_Q) \textbf{; Insert}(\textit{child}, \textit{e}_p, \textit{P}_Q) \\
7. \hspace{2em} \textbf{Insert}(\textit{e}_p, \textit{e}_c, \textit{P}_Q) \textbf{; Insert}(\textit{e}_c <, \textit{op}, \textit{P}_Q)
\]

8. \textbf{return} \(F_Q\)

Figure 2.9: The partitioning algorithm.

\textbf{Iterative operators} which are operators where a instance of itself may have a outgoing edge connected to the input of another instance of itself. An example of a iterative operator is the AGGR-MERGE operator.

\textbf{Non specific operators} which are operators that are not sensitive to their placement, yet have different effects on the size of their output. Examples of such operators are SELECT and PROJECT.

The edge selection criteria are semantic, in the case of location- or attribute-sensitive operators because correctness criteria constrain placement, as well as pragmatic, in the case of an operator whose output size is larger than that of its child in which case placement seeks to reduce the overall network traffic through reduced intermediate results. Assume \textit{Size} to be a function that estimates the size of the output of an operator or fragment, or the total output size of a collection of operators or fragment siblings, as described in [15]. Figure 2.9 shows the algorithm that computes a FAF \(F_Q\) from a PAF \(P_Q\).

The fragment allocation algorithm in Figure 2.10 determines where in the RT to place a fragment, using heuristics that aim to reduce the size of intermediate
transmissions. The reasoning behind reducing the size of intermediate results is that tuple transmission is the most energy-expensive operation and, thereby, reducing the energy cost of transmitting results in slower depletion of energy stocks.

A confluence site is a notion used for placement of attribute-sensitive operators. A confluence site in this context is a site where more than two copies of the output of a given logical operator travels through. For example, given the NESTED_LOOP_JOIN operator within the PAF in Figure 2.6(b), a confluence site would be a site where all the input streams from copies of F3 and F1 travel through. In Figure 2.10, \( s \Delta \text{op} \) is true iff \( s \) is the deepest confluence site for \( \text{op} \).

Fragments that contain SELECT, PROJECT, JOIN or AGGREGATION operators are more flexible in where they can be located. Such fragments have different effects on the size of their output and therefore their positioning can have a significant effect on the overall energy cost of a QEP. For example, a fragment containing solely a SELECT operator can potentially filter all of its input. It is, therefore, preferable to attempt to place this fragment as close as possible to the acquisition nodes, i.e., as far upstream as possible with respect to the direction of data flow. Fragments that contain JOIN and AGGREGATION operators are attribute-sensitive and must be placed on confluence sites.

An AGGREGATION operator is unique, in the sense that, physically, it is broken down into three different operators (AGGR-INIT, AGGR-MERGE and AGGR-EVAL), each of which can be placed in different fragments. This is done to reduce the size of intermediate results, in the sense that AGGR-INIT and AGGR-MERGE operators can operate over a subset of input eagerly and incrementally reduce the size of the overall output for the logical operator. The notion of a confluence node does not apply to AGGR-INIT, which therefore, is not thereby constrained. For a fragment with an AGGR-MERGE operator, a confluence site
is one into which the outputs of two or more AGGR-INIT or AGGR-MERGE (due to its iterative nature) operators flow. For a fragment with a AGGR-EVAL operator a confluence site is a site to which output from all AGGR-MERGE operators flow. This means that the notion of a confluence site applies to both AGGR-MERGE and AGGR-EVAL operators. The most energy-efficient confluence site on which to place an operator is the closest to the acquisition nodes, as this is the earliest point where the operators can contribute to the reduction of the size of the output given the incoming tuples.

\begin{figure}[h]
\begin{verbatim}
FRAGMENT-INSTANCE-ASSIGNMENT(F_Q, R_Q, Size)
1  D_Q ← F_Q
2  \textbf{while}
3     \hspace{1em} \triangleright \text{post-order traversing } D_Q
4     \hspace{1em} \triangleright \text{let } f \text{ denote the current fragment}
5     \hspace{1em} \textbf{do if } op \in f \text{ And } op.\text{LocationSensitive } = \text{yes}
6     \hspace{2em} \textbf{then for each } s \in op.\text{Sites}
7     \hspace{3em} \textbf{Assign}(f.\text{New}, s, D_Q)
8     \hspace{1em} \textbf{elseif } op \in f \text{ And } op.\text{AttributeSensitive } = \text{yes}
9     \hspace{2em} \textbf{And } Size(f) < Size(f.\text{Children})
10    \hspace{2em} \textbf{then while}
11       \hspace{3em} \triangleright \text{post-order traversing } R_Q,
12       \hspace{3em} \triangleright \text{let } s \text{ denote the current site}
13       \hspace{4em} \textbf{do if } s \Delta op
14       \hspace{5em} \textbf{then Assign}(f.\text{New}, s, D_Q)
15       \hspace{4em} \textbf{elseif } Size(f) < Size(f.\text{Children})
16       \hspace{5em} \textbf{then for each } c \in f.\text{Children}
17       \hspace{6em} \textbf{do for each } s \in c.\text{Sites}
18       \hspace{7em} \textbf{do Assign}(f.\text{New}, s, D_Q)
19       \hspace{6em} \textbf{else Assign}(f.\text{New}, R_Q.\text{Root}, D_Q)
20    \hspace{1em} \textbf{else Assign}(f.\text{New}, R_Q.\text{Root}, D_Q)
21  \textbf{return } D_Q
\end{verbatim}
\end{figure}

Figure 2.10: The fragment allocation algorithm.

Communication between fragments is abstracted away through the use of EXCHANGE operators. An EXCHANGE operator in SNEE consists of up to three different physical operators (a consumer, a relay and a producer). A consumer
operator handles transmissions of tuples to the next hop, a producer operator handles receipt of tuples from the previous hop, and if the producer and consumer are more than one hop apart, a relay operator receives and transmits tuples with no processing of its own.

In the running example, the PAF in Figure 2.6(b) is converted into the DAF shown in Figure 2.11. The decisions which resulted in Figure 2.11 are as follows:

1. The ACQUIRE as well as the corresponding time-window for the FOREST extent are placed within the same fragment, as they can operate solely on each other’s input. As the ACQUIRE operator is location-sensitive, the fragment is placed on nodes 7, 3 and 4.

2. The ACQUIRE as well as the corresponding time-window for the FIELD extent are placed in the same fragment, as they also can operate solely on each other’s input, and is placed on nodes 5 and 1 since it is location-sensitive.

3. The JOIN operator is placed on the deepest confluence node through which tuples from nodes 7, 3, 4, 5 and 1 flow, viz., node 1. The JOIN operator is alone in a fragment.

4. The DELIVER operator is placed alone in a fragment, as it requires the output of the JOIN operator and yet cannot be co-located with it because it is location-sensitive and pinned to the base station, i.e., node 0.

5. Exchanges are placed between the fragments. In this case, they connect node 7 to node 8 through nodes 6 and 5; nodes 3 and 4 to node 8 through node 2; nodes 5 and 1; nodes 1 and 8; and, finally, nodes 8 and 0.

It is worth noting that by abstracting away the communication between fragments in the DAF, it is unclear exactly which operators are placed in which site,
or where EXCHANGE relays are placed. Therefore, the integration of the DAF with the RT is represented by an instance fragment tree (IFT) [34] that makes explicit which fragments run on each site, including all parts of all EXCHANGE operators. An example of an IFT that corresponds to the DAF in Figure 2.11, and the RT in Figure 2.7 is shown in Figure 2.12. The set of all fragments and EXCHANGE operators allocated to a particular execution site are referred to as the \textit{site task}.
Figure 2.12: Query in instance fragment tree form.
2.3.4.3 When-Scheduling

The when-scheduling stage takes a DAF and determines when each fragment is run at every cycle of the agenda, as well as when each site transmits outgoing tuples. This results in an agenda that avoids collisions in transmissions i.e., that implements a time-division multiplexing approach to medium access control. If the communication was not coordinated in a global manner (e.g., by some other kind of medium-access control protocol), transmission collisions could occur, which in turn would require a site to retransmit its data, using up more energy and delaying the delivery. These events might lead to a failure to meet QoS expectations. By deciding precisely and specifically when each transmission is to take place, SNEE can determine the maximum number of tuples a given site needs to transmit using estimation models. This means that the sites that participate in communication only need to be turned on for the minimal amount of time required to actually transmit (or receive) the maximum amount of tuples it generates (or expects). This reduces the amount of energy used by the sites.

In the process of deciding on an agenda, SNEE applies another energy-saving technique viz., buffering tuples in the sites’s and only transmitting them in batches later. This saves energy as long bursts, if they are rare, are more energy-efficient than short bursts that are frequent, as there is less overhead from turning the radio on and off in the former than in the latter. In the when-scheduling stage, the amount of buffering ($\beta$) is maximised for all sites whilst still ensuring that the delivery time QoS specified by the query is met and that every site can buffer the tuples. Buffering is executed by avoiding transmitting tuples for a number (referred to as the buffering factor) of acquisition intervals. For example, assume an application with the QoS expectations that motes will sense the environment every hour and the application will delivery results every 24h. This application
When-Scheduling($D_Q$, $R_Q$, $\alpha$, $\delta$, Memory)

1. while
   1.1 pre-order traversing $R_Q$,
   1.2 let $s$ denote the current site
2. do $reqMem_e \leftarrow reqMem_f \leftarrow 0$
3. for each $f \in s$.AssignedFragments
5. do $x \leftarrow Memory(f,EXCHANGE)$
6. $reqMem_f \leftarrow + Memory(f) - x$
6. $reqMem_e \leftarrow + x$
7. $\beta^*[s] \leftarrow \lfloor \frac{s.AvailableMemory - reqMem_f}{reqMem_e} \rfloor$
8. $\beta \leftarrow min(\beta^*)$
9. while agenda.DeliveryTime > $min(\alpha * \beta, \delta)$
10. decr($\beta$)
11. return $\beta$

Figure 2.13: Computing an agenda buffering factor.

could potentially buffer 23 readings before having to transmit the block of data to the end user. This would result in a buffering factor between 0 and 23 given the memory constraints of the sites to which the application was planned to execute on. The algorithm that calculates the buffering factor is shown in Figure 2.13 where the function $memory$ uses cost estimation models defined in [16] to estimate the memory required by a fragment. Note, that for continuous execution for queries, the agenda just repeats.

The agenda can be visualised as a matrix, where rows denote points in time in the query evaluation cycle, columns denote participating sites, and the content of each cell defines the task scheduled for that node at that time. The agenda for the example query in Figure 2.4(b) is shown in Figure 2.15, where fragments are identified by their unique identifier as in Figure 2.11, with subscripts denoting the buffered instance; the notation $txn$ (resp., $rxn$) denotes that site at that time is transmitting to (resp., receiving from) site $s$; a row labelled sleeping denotes the fact that, for that slot, all the sites in the WSN are in the low power state. The
algorithm that generates the agenda is shown in Figure 2.14, where \textit{duration} is a function that uses cost estimation models defined in [16] to estimate the length of time a fragment takes to compute.

As has already been remarked, in being governed by an agenda, a SNEE QEP implements a simple form of TDMA (time-division multiple-access) to channels which is often economical, provided that the estimation models are accurate (and [16] shows that the ones used in SNEE are). Any changes to the timing of the operators or transmissions requires the agenda to be recomputed and hence the QEP to be recompiled and propagated into the WSN.

2.3.4.4 Code Generation

The final step in the SNEE compilation/optimisation stack takes the RT, IFT, and agenda generated in the multi-site optimisation stage and produces source files (for each participating site in the QEP). The code generator is currently designed to generate nesC [41] code for execution in the TinyOS [58] environment. The site tasks for each node are translated into nesC code that embodies the computing and communication requirements depicted in abstract form by diagrams like the one shown in Figure 2.16. The figure describes the activity in site 8 where tuples are received from the three input sites 2, 5 and 1, processed and the output transmitted onwards to node 0. In the figure: arrows denote component interaction, square-cornered boxes denote software abstractions of hardware components, dashed, round-cornered boxes denote components that carry out agenda tasks in response to a clock event, and finally ovals denote operators in a given fragment. In Figure 2.16 tuples received from sites 5 and 1 that were generated by instances of Fragment F1 are placed in the \textit{F1 output tray}. Conversely tuples received from sites 5 and 2 that were generated by instances of Fragment F2 are placed in the \textit{F2 output tray}. Fragment F3 consumes the tuples from the trays.
2.3. SNEE SNQP

\text{Build-Agenda}(D_Q, R_Q, \alpha, \beta, \text{Duration})

▷ schedule leaf fragments first
1 \hspace{1em} \text{for } i \leftarrow 1 \text{ to } \beta
2 \hspace{1em} \text{do for each } s \in R_Q.\text{Sites}
3 \hspace{2em} \text{do } \text{nextSlot}[s] \leftarrow \alpha \ast (i - 1)
4 \hspace{2em} \text{while}
5 \hspace{3em} \text{▷ post-order traversing } D_Q
6 \hspace{3em} \text{▷ let } f \text{ denote the current fragment}
7 \hspace{4em} \text{do if } f.\text{IsLeaf} = \text{yes}
8 \hspace{5em} \text{then } s.f.\text{ActAt} \leftarrow \left[ \right]
9 \hspace{5em} \text{for each } s \in f.\text{Sites}
10 \hspace{6em} \text{do } s.f.\text{ActAt}.\text{Append} \text{nextSlot}[s]
11 \hspace{5em} \text{nextSlot}[s] \leftarrow + \text{Duration}(s.f)

▷ schedule non-leaf fragments next
12 \hspace{2em} \text{while}
13 \hspace{3em} \text▷ post-order traversing } R_Q,
14 \hspace{3em} \text▷ let } s \text{ denote the current site
15 \hspace{4em} \text{do while}
16 \hspace{5em} \text▷ post-order traversing } D_Q
17 \hspace{5em} \text▷ let } f \text{ denote the current fragment
18 \hspace{6em} \text{do if } f \in s.\text{AssignedFragments}
19 \hspace{7em} \text{then } f.\text{ActAt} \leftarrow \text{nextSlot}[s]
20 \hspace{7em} \text{nextSlot}[s] \leftarrow + \text{Duration}(f)

▷ schedule comms between fragments
21 \hspace{4em} s.\text{TX}.\text{ActAt} \leftarrow \max(\text{nextSlot}[s], \text{nextSlot}[s.\text{Parent}])
22 \hspace{4em} s.\text{Parent}.\text{RX}(s).\text{ActAt} \leftarrow s.\text{TX}.\text{ActAt}
23 \hspace{4em} \text{nextSlot}[s] \leftarrow + \text{Duration}(s.\text{TX})
24 \hspace{4em} \text{nextSlot}[s.\text{Parent}] \leftarrow + s.\text{Parent}.\text{RX}
25 \hspace{2em} \text{return } \text{agenda}

Figure 2.14: The agenda construction algorithm.
through its consumers, and executes the join operator. The output from the join operator is fetched by the producer which writes them in the \( F3 \) output tray. When the site is scheduled to transmits its outputs the radio reads in tuples from the \( F3 \) output tray and transmits them.

Figure 2.16: TinyOS component diagram for site 8 in Figure 2.12.
2.4 Summary

End users have, over time, desired data from ever more diverse applications and locations. From a single data store on a single central server to stores distributed over a large geographical area, in more recent cases also combining streamed data that cannot be stored for future processing. Therefore, the kinds of query processing approaches required by modern applications had been expanding. Each approach has resulted in challenges to the community to ensure that the data is provided whilst meeting end-user QoS expectations. In the context of SNQP, much research has been done in terms of extending the lifetime of the deployment through energy and memory saving techniques.

However, the computational platforms characterised by mote-level WSNs are more unstable than others and therefore give rise to new extra challenges in regard to meeting QoS expectations. Specifically, resilience to failure events remains a relatively under-researched topic in the context of SNQPs, even though WSN are an unstable fabric, as the nodes and edges are prone to failure due to energy depletion, hardware malfunction and hostile deployment conditions. In the context of SNEE, changes in the physical fabric of the deployment can have significant negative effects: from producing incomplete results to failure to produce any results. Motivated by this need in general, and more specially, by the lack of resilience in SNEE (as one of the most advanced SNQPs of its kind) this thesis describes and evaluates adaptive techniques that endow SNEE generated QEPs with resilience to different types of changes in the physical fabric of the
SNEE was selected as our starting point for implementing the adaptive techniques contributed in this thesis for two reasons: firstly, the lack of adaptive capabilities, thereby allowing us a clean benchmark to compare the effectiveness of any of our techniques, and secondly, the predictability of its behaviour in a static unchanging environment, when other SNQPs, even under ideal conditions, are prone to unpredictable behaviours in terms of how well they meet the QoS expectations.
Chapter 3

Resilience to Unpredictable Node Events

This chapter presents two techniques designed to prolong the lifetime of a query $Q$, executing on a deployment $D$, when nodes in $D$ can fail unpredictably. The SNEE SNQP follows a compile/optimise once, execute many times paradigm for its QEPs, therefore there is no consideration of the effect that unpredictable node failure events would have on the operations of the executing QEP. Furthermore, there is no consideration of how to go about recovering from such events when they occur.

In the context of this thesis, unpredictable node failure events are considered permanent node failures which occur due to events that cannot be predicted with analytical models e.g. if an animal stands on a mote and destroys it.

The site tasks generated by the SNEE SNQP are strictly defined in terms of what fragments are executed on each site, and when the fragments are executed. This means that the effect of an unpredictable node failure event experienced by a participating node is that the QEP either generates incomplete results, or crashes completely. Therefore, adapting to unpredictable node failure events can extend
the functional lifetime of a query $Q$, running on a deployment $D$, from the time when the first *unpredictable node failure event* occurs, to the time at which the last QEP of an sequence of functionally equivalent QEPs $[q_i, ..., q_n]$ fails. The QEPs in the sequence are generated and switched to when some node $n_i$ experiences a node failure event $e_i$ and where $q_{i+1}$ compensates for the consequences of $e_i$.

This chapter covers two distinct techniques that, over the lifetime of a query, will generate a sequence of functionally equivalent QEPs. Each technique generates a new QEP $q_i$ that is optimised in a different way to compensate for the effect of an *unpredictable node failure event* $e$. The first technique is referred to as the *complete re-optimisation strategy*. It aims at generating a replacement QEP $q_{i+1}$ that is optimised for the new deployment layout after $e_i$ has occurred. The second technique, referred to as the *regional re-optimisation strategy*, aims to generate a replacement QEP $q_{i+1}$ that retains as much of $q_i$’s logical and physical structure as possible. This has the potential to reduce the amount of energy used in the adaptation process and also to enable faster adaptations.

Definitions and explanations of techniques that either preventively change a QEP ahead of *predictable node failure events* or that adapt to *unpredictable failure events* in resiliently designed deployments, are described in later chapters, as follows. Chapter 4 presents a technique that plans a sequence of QEP transitions at compile time, with the overall aim of circumventing preventable *predictable node failure events*. Chapter 5 presents a general framework that can handle both types of *unpredictable failure events* through the exploitation of resiliently designed deployments. In this chapter, the focus is on techniques that can be integrated into the SNEE SNQP without intensive redesign and that adapt to *unpredictable node failure events* on deployments with exploitable redundancy.

Note that neither of the techniques described in this chapter can adapt to *unpredictable node failure events* that affect acquisition nodes. This is because
there is no concept of a redundant sensor node for either strategy, and therefore it is impossible to generate a functionally equivalent QEP for the new deployment state where there are fewer source nodes. The technique described in Chapter 5 on the other hand has the ability to replace failed acquisition nodes.

3.1 Overview

*Unpredictable node failure events* are, by their nature, impossible to model accurately. This means that techniques targeting *unpredictable node failure events* are always reactive in nature, as they can only act when an event occurs at runtime. Any reactive strategy changes the paradigm, from compile/optimise once, execute many times, to a compile/optimise many times, execute many times paradigm.

In the context of SNQP, adapting to a *unpredictable node failure event* can be broken down into three separate sub-problems:

- the detection of an *unpredictable node failure event* in a timely fashion;
- the generation of a new QEP that takes into account the changes in the deployment fabric; and
- the shipping of the new QEP onto the network in an energy-efficient manner.

To detect an *unpredictable node failure event*, there is a need to monitor the deployment $D$ in such a way as to be able to determine when such an event has occurred and, more specifically, which node has experienced the event. Once an event has been detected and identified, a new QEP needs to be generated that conforms to some adaptive strategy. Once the new QEP $q_{i+1}$ has been generated, it is assessed on how to migrate from the QEP $q_i$ that is currently executing on $D$ to the new $q_{i+1}$ (this process is referred to as an *adaptation*) whilst avoiding excessive expenditure of energy and time.
Figure 3.1 shows how the original SNEE infrastructure (already presented in Figure 2.3) can be modified to allow a passive monitoring procedure (defined by Stages 2.1, 2.2 and 2.3 in Figure 3.1) that detects unpredictable node failure events and an adaptive technique that adapts to the events. To allow this infrastructure to automatically adapt to events, a simple monitor-assess-plan-execute (MAPE) control loop [85] has been implemented as follows:

M  A monitoring stage collects information about the executing QEP. In this case, M is represented by Stages 2.1 and 2.2 in Figure 3.1 where the output of the executing QEP \( q_i \) is monitored.

A  An assessment stage determines if any events have occurred that need an adaptive response. In this case, A is represented by Stage 2.3 in Figure 3.1,
3.1. OVERVIEW

where the detection of an *unpredictable node failure event* \( e \) is handled.

**P** A planning stage determines how to change the QEP \( q_i \) being monitored. In this case, **P** is represented by Stages 2.3 and 2.4 in Figure 3.1, where Stage 2.3 calls a planner that selects which strategy to execute to generate \( q_{i+1} \), and Stage 2.4 determines how to ship \( q_{i+1} \) onto \( D \) cost-effectively.

**E** An execution stage executes the changes from \( q_i \) to \( q_{i+1} \). In this case, **E** is represented by Stage 2.5 in Figure 3.1, where the shipping of \( q_{i+1} \) takes place.

When this MAPE control loop is added to **SNEE**, it gives rise to an adaptive version of **SNEE**, referred to as **ADSNEE**, which reactively adapts to runtime events.

### 3.1.1 Monitoring Technique

Stages 2.1, 2.2 and 2.3 in Figure 3.1 represent the monitoring technique used in **ADSNEE**. The main reason for selecting a centralised, approach for detecting *unpredictable node failure events*, is its low energy footprint. For each *site task* in the executing QEP, it is possible to estimate how many tuples are transmitted from the site. Assuming regular updates of the percentage of tuples that pass each operator, it is possible to predict the number of tuples that should arrive at the base station for a given agenda cycle (assuming an ideal environment) through the use of a cardinality estimation model, as shown in Figure 3.1. A more detailed description of the **ADSNEE** cardinality estimation model can be found in Appendix A.

If the estimated cardinality differs significantly from the actual number of tuples delivered, then **ADSNEE** assumes that a participating node has failed.


LocaliseNodeFailure(QEP qep, Environment e, ETC, tupleDifference)
1. possibleFailedNodes ← []
2. for each n in qep do
3.   NETC ← cardinalityEstimationModel.estimateTuples(n, qep)
4.   if NETC ≈ ETC - tupleDifference then
5.     possibleFailedNodes ← possibleFailedNodes ∪ n
6. if possibleFailedNodes.size() == 1 then
7.   return possibleFailedNodes.get(0)
8. if possibleFailedNodes.size() == 0 then
9.   sendAliveRequestsToNodes(qep.allNodes())
10. if possibleFailedNodes.size() > 1 then
11.   sendAliveRequestsToNodes(possibleFailedNodes)
12. while possibleFailedNodes.size() > 1 or passedTimeout do
13.   message ← receivedAliveResponse()
14.   possibleFailedNodes.remove(message.From())
15. return possibleFailedNodes

Figure 3.2: Pseudo code for determining which node has experienced a unpredictable node failure event.

Figure 3.2 shows the algorithm that determines which node has experienced the unpredictable node failure event. The algorithm cycles through the nodes that, when failed, could have an effect on the cardinality, and re-runs the cardinality model to recalculate the estimated output for the QEP with each primary-failed node fn. If the estimated result is the same as the actual received result, it stores fn in a array of possible failures possibleNodeFailures. If more than one node can exhibit the same effect on the QEP, then alive request messages are sent to the nodes listed, as possibleNodeFailures in an attempt to determine which have failed. If there are no nodes that the model has flagged as potential causes of the observed effect, then alive request messages are sent to all the participating nodes. This is represented by Stage 2.3 in Figure 3.1.

Note that the cardinality estimation model requires up-to-date selectivity measurements for each QEP fragment, so that it can estimate accurately the
number of output tuples emitted by the fragment. If such selectivity updates are too frequent, the number of messages would increase, and have a detrimental effect on the lifetime of the query. Other node-failure detection techniques in the literature [80, 101, 61] use neighbouring nodes to monitor communications. These techniques often detect a failure by tracking the period since they last have heard from a neighbour. If a node has not been heard from within a given time period, a neighbour either assumes it has failed or sends an alive request message and waits a given period for a response. Each of these strategies are highly distributed in nature and require energy to be used through the transmission of alive request messages and alive response messages on a regular basis as well as through the monitoring procedure. The strategy used here, in comparison, minimises resource to alive requests/response messages and uses update messages to report changes in selectivity, which need not be very frequent.

Note that, whichever strategy is used, the overhead of detecting an unpredictable node failure event would be identical for any of the techniques discussed in this thesis, and therefore this cost is omitted from the evaluations of the reported contributions. The effect of the monitoring technique on the lifetime of $Q$, is considered out of scope of this thesis, but is, of course, a configurable parameter.

The next two sections are devoted to the description of the techniques used to adapt a running QEP $q_i$ when a unpredictable node failure event has been detected.

### 3.2 Complete Re-optimisation

The first technique is the complete re-optimisation strategy which seeks to generate the QEP that has the smallest energy footprint when executing on the new state of the deployment, i.e., after the unpredictable node failure event. This
strategy uses all available information about the network and re-optimises over the entire network.

3.2.1 Motivation

As the SNEE compilation stack generates energy- and memory-efficient QEPs, a case can be made that by calculating a new optimised QEP for the new deployment state, the lifetime of the deployment could be extended in comparison with state-of-the-art SNQPs.

By not considering what actions are needed to change between $q_i$ and $q_{i+1}$, there is the potential that a $q_{i+1}$ would result in large amounts of re-programming of nodes in $D$ during an adaptation. Reprogramming a node $n$ in $D$ is costly in both energy and time, as it requires the per node transmission of binary images to the participating node in $D$ through multi-hop communication. This can result in thousands of packets being transmitted. As mentioned previously, packet transmission is the most energy-expensive operation in mote level WSN deployments. In terms of time, writing a binary image to the on-board flash memory is the slowest operation in mote-level WSN deployments.

The overall reasoning behind the complete re-optimisation technique is that the extra energy and time used during the adaptation process may be compensated for by the optimised energy footprint and extended lifetime of $q_{i+1}$.

3.2.2 Execution Stack

The complete re-optimisation strategy is the simplest of the two presented in this chapter, as the first three stages of its execution stack in Figure 3.3 are identical to the multi-site phase of the SNEE compilation stack in Figure 2.5. At the end of Stage 3, a new RT, DAF and Agenda will have been computed, that together
3.2. COMPLETE RE-OPTIMISATION

Figure 3.3: Complete re-optimisation execution stack.

determine \( q_{i+1} \).

Assume the example IFT from Section 2.3, referred to from now on as \( q_i \), overlaid on the deployment, described in Section 2.3 and depicted in Figure 3.4(a). Now assume that Node 8 experiences an *unpredictable node failure event*. The IFT generated by the complete re-optimisation strategy, referred to as \( q_{i+1} \), is depicted in Figure 3.4(b) and is discussed in more detail below.

### 3.2.3 Adaptation Break Down

Stage 4 of Figure 3.3 compares the structure of \( q_i \) and \( q_{i+1} \) and determines how to migrate between \( q_i \) and \( q_{i+1} \) cost-effectively. There are five possible basic (or primitive) *adaptation actions* on nodes in \( D \). Migrating between \( q_i \) and \( q_{i+1} \) makes use of combinations of these basic actions. The goal is to select the combination that is most cost-effective. The basic actions are:

1. reprogramming the node completely (installing a fresh binary for it to execute);
2. redirecting the destination to which the outgoing tuples from the node are sent;

3. deactivating the node so that it no longer participates in the QEP;

4. activating the node so that it starts participating in the QEP; and

5. shifting the node’s agenda by a specific time period, thereby changing when the node executes its QEP fragments and engages in data transport.

In the example, to migrate between $q_i$ (shown in Figure 3.4(a)) and $q_{i+1}$ (shown in Figure 3.4(b)) a combination of the adaptation actions are required. These are:

- The reprogramming of Nodes 5 and 2 to remove relay operators.

- The reprogramming of Nodes 1 and 9 to add additional operators.
• The deactivation of Node 6, as it is no longer to participate in QEP execution.

• The redirection of Node 2’s outgoing tuples to Node 1 instead of the failed Node 8.

• The redirection of Node 4’s outgoing tuples to Node 9 from Node 2.

• The redirection of Node 5’s outgoing tuples to Node 9 from the failed Node 8.

• The change to Node 4’s agenda by an offset to take into account that new scheduled transmission period.

These actions are then carried out in the adaptation execution stage, where pre-defined messages are forwarded from the base station to the nodes in question. For reprogramming, the messages contain the new binary image, which is sent down $q_{i+1}$’s RT to the node in question, where it is written onto flash memory. For redirection, activation, deactivation and temporal adjustment, one message is sent communicating changes in variables in the binary, e.g., changing the node id that acts as its parent id for redirection purposes.

### 3.3 Regional Re-optimisation

The second technique is the *regional re-optimisation strategy*. It aims to generate a $q_{i+1}$ that requires less expenditure of energy during the adaptation process than the *complete re-optimisation*. The strategy uses all available information about the network and, in the worst case, can result in changes over the entire network, but, more often that not, repairs to $q_i$ are likely to cause smaller disruptions to the structure of $q_i$. 
CHAPTER 3. RESILIENCE TO UNPREDICTABLE NODE EVENTS

3.3.1 Motivation

The reasoning behind the *regional re-optimisation strategy* is that by avoiding unnecessary reprogramming of nodes during adaptations, each adaptation itself will be cheaper in terms of both time and energy metrics. In comparison to the *complete re-optimisation strategy*, the *regional re-optimisation strategy* would be applicable to different types of applications as it incurs less expenditure of time and energy.

For example, in the Volcano case study [120], a node fn can experience an *unpredictable node failure event e* due to the volcano erupting. When e occurs, adapting to e as quickly as possible is desirable, as the period of time just after e would contain the most interesting results for the end user. Therefore, the period of time in which Q and D are down adapting to e needs to be as short as possible. The potential extended lifetime of the deployment would be less important once e has occurred, as it is possible that the entire deployment would be destroyed by the eruption.

The downside to the *regional re-optimisation strategy* is that the QEPs generated have the potential to have a higher energy footprint for normal execution than a QEP generated by the *complete re-optimisation strategy*. This could result in the *regional re-optimisation strategy* QEPs producing a shorter functional lifetime overall.

3.3.2 Execution stack

The regional strategy works in five stages as depicted in the *regional re-optimisation strategy* execution stack in Figure 3.5. These stages can be summarised as follows:

**Stage 1.0** aims to reduce the number of nodes that are available for reprogramming;
Stage 2.1 attempts to repair the RT of the deployed QEP with the available reprogrammable nodes;

Stage 2.2 extends the set of available reprogrammable nodes by one node and retries stage 2.1, if it is not possible to repair the deployed QEP’s RT with the available reprogrammable nodes;

Stage 3.0 moves any fragments that were allocated to the failed node onto nodes in the new RT, whilst aiming to reduce reprogramming as much as possible;

Stage 4.0 invokes the (original) SNEE when-scheduler, as described in Section 2.3.4.3; and

Stage 5.0 executes the same adaptation breakdown process as described in Section 3.2.3.

In the following subsections, a detailed explanation is given of how each step operates using the example in Figure 3.4(a).

3.3.2.1 Node Pinning

The pinning stage (Stage 1 in Figure 3.5) aims to constrain the routing-tree construction algorithm, by removing participating nodes from consideration in the connectivity graph. The goal is to avoid long-range disruption as might ensue from the consideration of nodes that already participate in the QEP. This is done by traversing the $q_i$’s RT (overlaid on the connectivity graph in Figure 3.6(a)) and each node apart from the failed node $fn$ is defined as a pinned node. The rest of the nodes in the connectivity graph are defined as reprogrammable. $q_i$’s RT is then updated to reflect the failure of $fn$, resulting in a disconnection in $q_i$’s RT as the children on $fn$ no longer have a parent to transmit their tuples to.
The pinned nodes, \( fn \) and the reprogrammable set are then used as inputs for the routing stage. In the example in Figure 3.6(a), assume node 8 experiences an \textit{unpredictable node failure event} \( e \). Nodes 0, 1, 2, 3, 4, 5, 6 and 7 are pinned (represented by the purple colouring), and Node 8 is removed from the RT, resulting in the disconnected RT shown in Figure 3.6(b).

### 3.3.2.2 Routing and Unpinning

Stage 2.1 in Figure 3.5 attempts to repair \( q_i \)’s RT. This is done by first locating sections of the RT around \( fn \) and considering any unpinned nodes (initially an empty set) as candidates for repairing the consequences of the failure. Each section is repaired independently. In the example, only one section is located, viz., Node 8.

Each section then gives rise to a \textit{Steiner tree} [59] problem, where the inputs are the leaf nodes in the section (Nodes 2, 5 and 1 in this case) and the root
in the section (Node 0) represented as double circled nodes in Figure 3.8. The heuristic algorithm from [32] is used over the union of the reprogrammable and unpinned nodes to compute the Steiner tree. If a Steiner tree is formed, then it is joined to the remainder of the disconnected RT to create a new complete RT. If the heuristic algorithm fails to find a tree to connect the mandatory nodes in a section, then Stage 2.2 is initiated. In this example, with the initial scope represented by Figure 3.6(b), no Steiner tree solution can be generated between Nodes 2, 1, 5 and 0, with only Node 9 available for reprogramming. This is due to Node 2 not being able to communicate directly with Node 9. Therefore, Stage 2.2 in Figure 3.5 is initiated.

Stage 2.2 expands the scope of the repair process by *unpinning* a previously pinned node. There are two heuristics for selecting a node for *unpinning*. These, in ranked order, are:

1. Select the parent of the section to be unpinned;

2. If no parents exist (i.e., we have reached the base station), randomly select
CHAPTER 3. RESILIENCE TO UNPREDICTABLE NODE EVENTS

(a) Incomplete RT with first adjusted scope
(b) Incomplete RT with second adjusted scope
(c) Repaired RT

Figure 3.7: Representation of the different scoped incomplete and repaired RT’s.

The reasoning behind these heuristics is that it is likely that the *unpinned* node will require reprogramming and, therefore, being higher up the RT will reduce the energy footprint of reprogramming by reducing the number of hops the binary carrying packets are required to traverse during the adaptation process. If there is no available parent then a random selection gives no bias to any particular node.

In the example, Node 0 would be chosen to be unpinned first and Stage 2.1 in Figure 3.5 would be initiated. In this case, there still would not be enough scope to repair the RT, as the unpinning process has not added any new links to the scope, as presented in Figure 3.7(a). Stage 2.2 in Figure 3.5 would be reinitialised and a random node selected. In this case, assume Node 1 was randomly selected to be unpinned and Stage 2.1 in Figure 3.5 is reinitialised again with the new scope (represented in Figure 3.7(b)). At this point, the heuristic algorithm can join Nodes 2, 5, 1 and 0 through the reprogramming of Node 1, resulting in the repaired RT in Figure 3.7(c).
3.3. REGIONAL RE-OPTIMISATION

(a) Original QEP illustrated in Figure 2.12.
(b) QEP generated by the regional re-optimisation strategy.

Figure 3.8: Representation of the original and regional generated QEPs.

3.3.2.3 Where Scheduling

Stage 3 takes the new RT and generates a DAF by placing all non-attribute sensitive, pinned QEP fragments, onto the pinned nodes. Any remaining QEP fragments (in this case, the one containing the join operator that was originally placed on Node 8) are placed on the most applicable reprogrammable site (Node 1 in this case), determined by the where scheduling phase discussed in Section 2.3.4.3. This results in the DAF shown in Figure 3.8(b).

3.3.2.4 When Scheduling and Adaptation Break Down

Stage 4 is the same as in SNEE, i.e., the SNEE when scheduler in Figure 2.3.4.3 is invoked, and the same comparison stage as mentioned in the complete re-optimisation strategy is executed. The comparison of $q_i$, represented in Figure 3.8(a), with $q_{i+1}$, represented in Figure 3.8(b), will result in the reprogramming of Node 1 and the redirection of tuples from Nodes 2 and 5 to Node 1. This
adaptation is guaranteed to be shorter in execution time that the complete re-optimisation strategy, due to only requiring the reprogramming of one node instead of four.

3.4 Evaluation

The experiments reported in Section 3.4.2.1 investigate whether the strategies described in Sections 3.2 and 3.3 can increase the functional lifetime of the deployment when faced with a sequence of unpredictable node failures events. The experiments in Section 3.4.2.4 investigate the time cost, and in Section 3.4.2.5 the energy cost, of the adaptations generated by the two strategies with a view to determining the circumstances in which each strategy may be suitable.

3.4.1 Experimental Design

In the experimental design, the following assumptions are made:

- The metadata has been collected. Metadata collection costs are modest compared with the evaluation of queries over many agenda cycles [98].
- Communication between nodes is reliable and stable.
- The probability of a tuple passing any selectivity constraint in the operators is 100%, i.e., selectivity factors are set to the worst case in all cases.
- Each strategy is executed on a host machine that has sufficient processing and memory capabilities to execute the adaptive strategies.
- The base station has a connection to the mains power supply (e.g., that it is directly connected to a mains-powered laptop by a USB cable that the mote can power itself with).
The experiments are based on 60 WSN topologies that were algorithmically generated, with corresponding physical and logical schemas. Of these, 30 topologies contain 30 nodes, a scale that is representative of typical environmental monitoring applications [60, 120, 84, 23], and are referred to as the small set. The other 30 contain 100 nodes, a scale that is representative of a large application [103, 82, 23] and are referred to as the large set.

For both sets, a third of the topologies were randomly selected to execute a SELECT * query; a second third of the topologies were randomly selected to execute an aggregation query, and finally, the remaining third of the topologies were selected to execute a window-based join query.

Each query was compiled and executed with three different sets of QoS expectations, as shown in Table 3.1.

- QoS 1 is representative of an application where the deployment spends most of its time acquiring data from the sensors and transmitting the data to the end user. Therefore, QoS set 1 is more focused on getting large quantities of data over a short period of time than on ensuring long functional lifetimes. The sniper localisation deployment [103] is an example where such QoS expectations hold.

- QoS 2 is representative of an application where the deployment trades off acquiring data at a faster rate for a longer lifetime. The Great Duck Island [82] and the Zebra deployments [127] are examples where these QoS expectations hold.

- QoS 3 is representative of an application where the deployment is entirely focused on saving energy by sensing the environment sparingly and delivering the data to the end user on a scale of hours. The glacier monitoring deployment [84] is an example where these QoS expectations hold.
CHAPTER 3. RESILIENCE TO UNPREDICTABLE NODE EVENTS

<table>
<thead>
<tr>
<th>QoS Parameter</th>
<th>QoS 1</th>
<th>QoS 2</th>
<th>QoS 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivery Interval</td>
<td>600 s</td>
<td>600 s</td>
<td>30000 s</td>
</tr>
<tr>
<td>Acquisition Interval</td>
<td>10 s</td>
<td>300 s</td>
<td>300 s</td>
</tr>
</tbody>
</table>

Table 3.1: QoSs expectations used.

3.4.2 Experimental Results

This section investigates whether the use of adaptive strategies can increase the lifetime of a query $Q$, running on a deployment $D$, thereby improving its BFB value when faced with several unpredictable node failure events. The investigation reports on four properties, as follows:

1. The estimated functional lifetime of the deployment.

2. The estimated number of tuples delivered to the end user over the lifetime of the deployment.

3. The amount of down time the application must endure because of the duration of the adaptation process.

4. The amount of energy used by the adaptation process.

Investigating the effect that several node failures has on the execution of a QEP, is motivated by the observation that, in real world deployments, it is likely that more than one node will fail over the lifetime of a query. The effect of a single unpredictable node failure event is studied in Appendix C. Due to the lack of widely-accepted, published node-failure models for real deployments, the decision was made to spread the unpredictable node failure event uniformly over the estimated QEP lifetime.
3.4. EVALUATION

3.4.2.1 Lifetime

To estimate the lifetime of a query $Q$ executing on a deployment $D$, a QEP $q_i$ must first be generated by inputting $Q$, the topology, physical and logical schemas into the ADSNEE compilation stack. Once $q_i$ is generated, it is shipped to the deployment and the SNEE energy cost model [16] (which is based on the AVRORA [116] energy model) is used to estimate the lifetime of each node in the deployment. The energy cost model takes into account:

- The energy expended from over-the-air-programming (OTAP) of $q_i$ onto the deployment.
- The initial energy levels of the individual motes deployed.
- The energy and time costs of adapting QEPs as described in Section 3.2.3.
- The energy and time costs of executing $[q_i, ..., q_n]$ upon $D$ estimated using the cost models in [16].

The lifetime of the node with the shortest time-to-failure is considered the deployment’s functional lifetime. In other words, it is assumed that once the first node fails the QEP will not be able to produce correct results (and it may fail completely). As already explained, a node is selected to fail and a new QEP is then generated using the strategy under evaluation and shipped to the deployment.

The process of selecting node $f_{n_i}$, to experience a unexpected node failure event $e_i$, is limited to:

- non-acquisition nodes that are confluence nodes as described in Section 2.3.4.2, or else,
• if no node meet the first requirement, then any non-acquisition node is selected.

The rationale for these restrictions are:

1. By only selecting non-acquisition nodes, the generated QEPs are functionally equivalent to their predecessors. Reactive strategies cannot adapt to the failure of an acquisition node because all acquisition nodes must contribute to the result at all times for the query results to be correct.

2. Confluence nodes have a significantly larger effect on the lifetime of a QEP than, say, an individual relay node, because confluence nodes often have more QEP fragments allocated to them and often result in hot spots in the execution.

Once $q_{i+1}$ has been shipped to $D$, the SNEE energy cost model [16] estimates the lifetime of the deployment with the remaining energy stocks. The number of unexpected node failure events, $fnc$, is then used to determine the number of agenda cycles that the new QEP, $q_{i+1}$, should complete before another node is selected to experience an unexpected node failure event, following a uniform distribution over the lifetime of the query. For example, assume that $fnc = 1$ and that $q_i$ is estimated to last 80,000 agenda cycles, then a node would be selected to experience an unexpected node failure event after 40,000 agenda cycles. If $fnc = 4$, then a node would be selected to experience an unexpected node failure event every 20,000 agenda cycles. If the strategy fails to adapt to the event $e_i$, then the estimated lifetime is defined as the point in time when $e_i$ occurred. If $fnc$ is reached, then the estimated lifetime is defined as the time period taken until the next node fails due to energy depletion. The logical flow diagram for this experiment can be found in Appendix G.1.1.
3.4. EVALUATION

(a) Estimated lifetime for all three strategies for a select * query over several unexpected node failure events for QoS set 1.

(b) Estimated lifetime for all three strategies for a select * query over several unexpected node failure events for QoS set 2.

Figure 3.9: Estimated lifetime for all three strategies for the select * query type over several unexpected node failure events.
CHAPTER 3. RESILIENCE TO UNPREDICTABLE NODE EVENTS

Figure 3.10: Estimated lifetime for all three strategies for an aggregation query type over several unexpected node failure events.

(a) Estimated lifetime for the aggregation query over several unexpected node failure events for QoS set 1.

(b) Estimated lifetime for the aggregation query over several unexpected node failure events for QoS set 2.
3.4. EVALUATION

(a) Estimated lifetime for all three strategies for a join query over several \textit{unexpected node failure events} for QoS set 1.

(b) Estimated lifetime for all three strategies for a join query over several \textit{unexpected node failure events} for QoS set 2.

Figure 3.11: Estimated lifetime for all three strategies for the join query type over several \textit{unexpected node failure events}. 
Figures 3.9, 3.10 and 3.11 present the estimated lifetimes of the large topology set for each query type, for QoS 1. The following can be observed:

- Both adaptive strategies significantly outperform the non-adaptive strategy by between 91.1% (after 1 event) and 1021.71% (after 8 events) in terms of the estimated functional lifetime of the query, when faced with unpredictable node failure events. This is because, in the non-adaptive case, once a node has failed, the query is considered to have failed completely.

- The effect of multiple adaptations on the estimated lifetime of the query for each adaptive strategy is variable (from -47.9% to 41.9%). Sometimes there is a small rise (e.g., of 5.27%) in lifetime performance, as seen in the first four node failures in Figure 3.9(a), for the regional re-optimisation strategy. This is because the adaptations have resulted in the reallocation of operators, from nodes that previously experienced a high workload to fresher nodes, which had either not participated in the QEP, or a lower workload in previous QEPs. This results in energy saving on the high-workload nodes and, therefore, in longer lifetimes of between 0.6% and 41.9% before a critical node fails from energy depletion, as seen in the last three node failures in Figure 3.11(a) or throughout the failures in Figure 3.10(a).

- Conversely, it is possible that the adaptations can result in the high-workload nodes being allocated even higher workloads. It is also possible that there is no change in workload, but, during the adaptation process, the high-workload nodes participate in other binary transmissions, thereby depleting their energy reserves e.g., the regional performance between 4 and 5 node failure events in figure 3.10(a).

- Due to this variable performance, neither adaptive strategy outperforms
the other on a regular basis, and therefore, there is no direct conclusion as to which strategy to use when aiming for the biggest boost to lifetime.

• The drop in functional lifetime of the join queries (between 34.0 % and 66.4 %), in comparison to the other two query types is due to joins giving rise to larger results by about 50 % in average transmitted, which thereby reduces the overall lifetime of the node on which it is placed by about 43.16% in average.

• The drop in benefit accrued between the adaptive strategies and the non adaptive strategy as one varies the QoS expectations, is due to the energy cost of the *site tasks* of the QEPs in QoS set 2 and 3 being dominated by the cost incurred by the motes being in the sleep state which is unavoidable and all nodes in D experience. This makes it difficult for the adaptive strategies to produce significant benefits.

Figures 3.12 and 3.13 present the estimated lifetimes for both topology sets for the QoS1 in Table 3.1. Figures 3.14 and 3.15 present the estimated lifetimes for both topology sets for QoS2 in Table 3.1. The following can be observed:
Figure 3.12: Estimated lifetime of both strategies in the face of several node failures for the small topology set with QoS set 1.

Figure 3.13: Estimated lifetime of both strategies in the face of several node failures for the large topology set with QoS set 1.
3.4. EVALUATION

Figure 3.14: Estimated lifetime of both strategies in the face of several node failures for the small topology set with QoS set 2.

Figure 3.15: Estimated lifetime of both strategies in the face of several node failures for the large topology set with QoS set 2.
• Neither strategy outperforms the other on a regular basis and therefore it is not clear how the estimated functional lifetime of the query can be used for differentiating between the two strategies. This could mean that both strategies need to be available for selection on every event.

• The lifetimes generated by the queries under QoS2 (Figures 3.14 and 3.15) are significantly larger than those generated under QoS1 (Figures 3.12 and 3.13) by between 55% and 60%. This is due to the energy cost of the QEP being dominated by (unavoidable) sleep state over the entire deployment lifetime, thereby allowing the QEP to execute for more agenda cycles (about 20% in average).

• The tighter clustering that can be observed in Figures 3.14 and 3.15 in relation to Figures 3.12 and 3.13 is also due to the QEP being dominated by the unavoidable sleep energy costs over the entire deployment lifetime as the effect results in the reduction of the benefits available from each strategy.

• Points within cluster A in Figures 3.12 and 3.14 represent node failures where the regional re-optimisation strategy was still able to adapt, whereas the complete re-optimisation strategy could not overcome the partitioning of the connectivity graph of $D$. This is not seen in the larger topology set, because there are more edges in the connectivity graph, and, therefore, more nodes would need to be lost, before a partitioning could occur.

To conclude, even in the face of several node failures, both adaptive strategies in ADSNEE outperform the non-adaptive version of SNEE in terms of lifetime. Neither adaptive strategy regularly outperforms the other, and, therefore, for an end user who is solely focused on increased lifetime, and therefore increased BFB
### Table 3.2: Average and standard deviation for each query type.

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Average lifetime</th>
<th>Standard Deviation in lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>star</td>
<td>58267</td>
<td>8285</td>
</tr>
<tr>
<td>Aggregation</td>
<td>68559</td>
<td>3045</td>
</tr>
<tr>
<td>Join</td>
<td>44246</td>
<td>4615</td>
</tr>
</tbody>
</table>

values, deciding which adaptive strategy to deploy is not obvious.

#### 3.4.2.2 Robustness Measurements

In Section 3.4.2.1, each topology in a given query set was run with the same query. To show that diverse queries grouped by type do not have a significant effect on the results, a robustness analysis was carried out. An average-performing topology for each query type was selected, and then randomly generated queries (presented in Appendix D) were executed on it. The average and standard deviation for each query type can be found in Table 4.5. Lifetimes were measured for both the adaptive and non-adaptive version of SNEE. Figure 3.16 presents the different lifetimes obtained from the various queries, and Figure 3.17 presents the difference between the strategies over various queries. The following can be observed:

- The effect on the overall lifetime of a deployment can be significantly different, depending on the query. This is mainly because of the number of packets that have to be transmitted, as defined by the time window, as well as the size of each tuple, as defined by the number of attributes each tuple contains.
Figure 3.16: The lifetime estimated for a given deployment over a collection of diverse queries.
Figure 3.17: The difference in lifetime between the different strategies over a collection of diverse queries.
• The overall effect of different forms of the same query type on the difference in performance between the adaptive and non-adaptive strategies is small in the case of aggregation and join queries (no larger than 2.92 %). The larger difference (up to 37 %) in the case of the select queries is due to the different number of attributes being requested from each query.

The conclusion from this experiment is that different queries of a particular query type have, overall, similar performance between the adaptive and non-adaptive version of SNEE and, therefore, for the rest of the experimental evaluation only one query is used per query type.

3.4.2.3 Number of Tuples Delivered

This section investigates the effect of a sequence of node failures on the number of tuples delivered to the end user, during query execution. Estimating the number of tuples delivered during the lifetime of a query $q_k$ is done in two stages. In the first stage, the number of tuples $tup$ that can be delivered per agenda cycle is estimated using the cardinality model described in Appendix A. The second stage multiplies $tup$ by the estimated lifetime of the QEP, as determined by the results in Section 3.4.2.1. The logical flow diagram for this experiment can be found in Appendix G.1.4.

Figures 3.18 and 3.19 present the estimated number of tuples delivered to the end user over the lifetime of the deployment for both topology sets with QoS1. Figures 3.20 and 3.21 present the estimated number of tuples delivered to the end user over the estimated lifetime of the deployment for both topology sets with QoS2. The following can be observed:

• As aggregation operators reduce all input tuples into a single output tuple, aggregation queries show no noticeable difference in total tuples delivered,
unless the entire data stream is disconnected. It is acknowledged that the accuracy of the resulting tuple could now be severely compromised.

- The effect of increasing the number of nodes within a deployment results in no discernible effect on the number of tuples delivered to the end user. The first reason for this is that there is a maximum limit on what both strategies can accomplish, which is that their functional lifetimes are capped to when the first acquisition node fails due to energy depletion. The second reason is that the same number of acquisition nodes is used in the larger topology set as in the small topology set.

- Points within cluster A in Figures 3.18 and 3.20 represent unexpected node failure events where the regional re-optimisation strategy was still able to generate QEP's that mitigated the failure, whereas the complete re-optimisation strategy was unable to adapt because of a partitioning of the connectivity graph. This is not observed in the larger topology set because there are potentially more edges that need to be lost before a partitioning occurs.

In conclusion, by adapting to several node failures during query execution, the number of tuples delivered to the end user can be increased beyond what is possible without adapting to these failures. As neither adaptive strategy outperforms the other on a regular basis with respect to the number of tuples delivered to the end user, selecting one strategy over the other is event-specific.
Figure 3.18: Estimated number of tuples delivered to the end user for the different topologies in the small topology set, given QoS expectation set one.

Figure 3.19: Estimated number of tuples delivered to the end user for the different topologies in the large topology set, given QoS expectation set one.
3.4. EVALUATION

Figure 3.20: Estimated number of tuples delivered to the end user for the different topologies in the small topology set, given QoS expectation set two.

Figure 3.21: Estimated number of tuples delivered to the end user for the different topologies in the large topology set, given QoS expectation set two.
3.4.2.4 Time Measurements

Time that is spent adapting to an unexpected node failure event is time that is not being spent in the production of query results. Therefore, it is important to consider adaptation time, as it will result in gaps in the result stream. This motivated the exploration of the total amount of time used in adaptations. The total time sums up the time used by each individual adaptation for each unexpected node failure event in a sequence of such events. The logical experimental flow diagram for this experiment can be found in Appendix G.1.2.

The effect of different QoS expectations on the time taken to adapt is minimal, because the QEP, and thereby the adaptation, is topology-specific and therefore the differences in adaptation time come mostly from the size of reprogrammable binaries. Furthermore, as the only difference in the binary resulting from QoS expectations is the memory requirements of the data buffers, which are relatively small compared to the total image size (often in the range of a few packets).

Figures 3.22 and 3.23 present the total time taken to adapt for both topology sets given QoS1. Figures 3.24 and 3.25 present the total time taken to adapt for both topology sets given QoS2. The following observations can be made:

1. In most cases the complete re-optimisation strategy produces adaptations which incur a significantly higher time cost than the regional re-optimisation strategy over several subsequent adaptations (between 4.0 % and 32.7 %). This means that using the complete re-optimisation strategy is likely to generate more downtime and therefore larger gaps in the output.
3.4. EVALUATION

Figure 3.22: Estimated time period taken up adapting to several node failures for the different topologies in the small topology set, given QoS expectation set one.

Figure 3.23: Estimated time period taken up adapting to several node failures for the different topologies in the large topology set, given QoS expectation set one.
Figure 3.24: Estimated time period taken up adapting to several node failures for the different topologies in the small topology set, given QoS expectation set two.

Figure 3.25: Estimated time period taken up adapting to several node failures for the different topologies in the large topology set, given QoS expectation set two.
2. The *regional re-optimisation strategy* on the other hand has variable performance, in that, most of the time, it initially produces adaptations that execute faster than the *complete re-optimisation strategy*, but, after a few *unpredictable node failure events*, the adaptations can take as long or longer than those generated by the *complete re-optimisation strategy*. In these cases, *unpredictable node failure events* are often located in the same region, thereby resulting in the *regional re-optimisation strategy* generating a communication hole in the deployment area. This results in the regional strategy having to execute larger adaptations to circumvent the communication hole.

In most agendas, there is a period of time at the end of an agenda cycle, where every node is sleeping (in Figure 2.15, this is after time point 3303364). When it is possible to send and process all the adaptation messages in the time between the start of the final sleeping period and the re-execution of the agenda (referred to as the *golden zone*), then it would be possible for the strategy to carry out the adaptation without exhibiting a gap in the result output stream. Formula 3.1 determines if an adaptation can exploit this *golden zone* by first summing the total time \( tt \) taken to: (a) calculate what adaptation messages are needed, (b) execute the adaptation in the deployment and (c) retransmit the lost tuples to the base station. Then, \( tt \) is compared to the length of the *golden zone* to see if the adaptation can fit inside the latter. The *golden zone* is defined in Formula 3.2 as the difference between the time at which the delivery operator is executed and the time at which the agenda repeats. It is worth noting that the size of the *golden zone* is directly affected by the number of participating nodes (due to the TDMA behaviour of the agenda, as explained in Section 2.3.4.3) and the acquisition interval defined for the query.
CHAPTER 3. RESILIENCE TO UNPREDICTABLE NODE EVENTS

goldenZone(agenda) > AdaptationTime(adaptationMessages) +
QEPgenerationTime() +
TimeToRetransmitLostResults() \quad (3.1)

goldenZone(agenda) = agenda.DeliveryTime() –
TimeWhenDeliverOperatorCompleted(agenda) \quad (3.2)

Figure 3.26: The tipping point for adaptations without disruption.

It is also worth noting that the AdaptationTime is the dominant cost in Formula 3.1, as both the QEPgenerationTime and TimeToRetransmitLostResults for either strategy is in the order of a few seconds, whereas the AdaptationTime is in the order of hundreds or thousands of seconds.

In conclusion, the time taken to adapt to a collection of unpredictable node failure events by the complete re-optimisation strategy is often larger than the time taken by the regional re-optimisation strategy. By exploiting the golden zone, the regional re-optimisation strategy may often be able to support faster acquisition rates whilst delivering uninterrupted output streams than is possible with the complete re-optimisation strategy. This means that end users who wish to have a greater likelihood of witnessing no discernible effect on the continuity of the result stream in unpredictable environments would prefer the regional re-optimisation strategy over the complete re-optimisation strategy.

3.4.2.5 Energy Measurements

During the adaptation process, messages are transmitted, resulting in additional extra energy drain, that could have a detrimental effect on deployment’s functional lifetime. The SNEE energy cost model [16] is used to estimate the total energy used up by the adaptation process, including an extension to estimate the
3.4. EVALUATION

energy used in writing the new binaries to flash. The logical flow diagram for this experiment can be found in Appendix G.1.3.

Figures 3.27 and 3.28 show the energy drain for both topology sets given QoS 1 over several adaptations. Figures 3.29 and 3.30 show the energy drain for both topology sets given QoS 2. The following can be observed:

1. In most cases, the complete re-optimisation strategy experiences significantly larger (between 9.0 % and 32.2 %) energy costs than the regional re-optimisation strategy. This is due to the larger number of nodes that need reprogramming in the adaptations generated, which results in more packets being transmitted.

2. There is a direct correlation between the length of time taken by an adaptation to complete and the amount of energy used during the process. This is because adaptations that require the transmission of many binaries, which takes time due to the relatively slow speeds achieved in writing to flash, also uses more energy due to the transmission of potentially thousands of messages containing the binary code.

In conclusion, even though the amount of energy used by each adaptive strategy can differ significantly (such amounts are equivalent to a few hundred agenda cycles), the energy cost is often spread over a large set of nodes and therefore has little effect on the query’s lifetime when compared with the energy savings that result from using previously inactive fresher nodes with larger residual energy stocks, in the new QEP.
CHAPTER 3. RESILIENCE TO UNPREDICTABLE NODE EVENTS

Figure 3.27: Estimated energy drain upon the deployment experienced from several adaptation events for topologies in the small topology set, with QoS set 1.

Figure 3.28: Estimated energy drain upon the deployment experienced from several adaptation events for different topologies and queries in the large set, with QoS set 1.
3.4. EVALUATION

Figure 3.29: Estimated energy drain upon the deployment experienced from several adaptation events for topologies in the small topology set, with QoS set 2.

Figure 3.30: Estimated energy drain upon the deployment experienced from several adaptation events for different topologies and queries in the large topology set, with QoS set 2.
3.5 Related Work

As discussed in Section 1.3, current SNQPs have little or no adaptive behaviour integrated into or around QEP runtime. A few SNQPs have limited support for unpredictable node failure events. TinyDB [80] and SmartCIS [77] both send the entire QEP onto every node in the deployment, and therefore they only need to consider the routing of tuples around a node that has experienced a unpredictable node failure event $e$ without needing to consider moving any in-network processing. Such systems can route tuples around the node that experienced $e$; each child node keeps track of when it last heard from its parent and if it surpasses a predetermined threshold, then it considers the parent to have experienced $e$ [79]. The child then selects a new parent by listening for the next closest parent node. SmartCIS [77] improves upon TinyDB with the particular goal of supporting resilience to unpredictable node failure events, through the introduction of multi-route transmissions. Once a node has experienced $e$, all query execution is stopped and a route discovery algorithm is engaged. The drawback of these approaches is that sending the QEP to every node results in more memory use on each node as well as a lack of overall communication discipline, leading to a greater risk of collisions and tuple loss though the use of load-shedding techniques which are required when there is no globally ensured agenda. Any adaptation made by these proposals is also based on local information and therefore may not lead to the best outcome for the deployment as a whole.

The QEPs generated by the AnduIN SNQP have no internal support for node failure, but unlike TinyDB, SNEE and SmartCIS which compile into TinyOS code [58], AnduIN compiles to Contiki [26] source code. The difference is relevant in our context because, unlike TinyOS, Contiki provides a TCP/IP-based
3.5. RELATED WORK

communication protocol stack. Thus, AnduIN benefits from the robust routing and transport properties built into TCP/IP. The draw-back is that TCP/IP requires much greater overheads in terms of energy and memory than the minimalistic, query-specific protocols used by TinyDB and SNEE. Some of the energy overheads stem from the need to maintain up-to-date connectivity paths as well as from the need to transmit acknowledgement packets. The memory overheads stem from having to store data structures, such as routing tables, which reduces the memory that can be allocated to the QEP. Therefore, there is a reduction in how much processing can be shipped to the WSN, and how much memory can be used for buffering and blocked transmission, which are, both, energy-saving features. AnduIN does not adapt to unexpected node failure events that affect acquisition nodes, which is also where in AnduIN’s case all the in-network processing takes place.

It is worth noting that in all the above mentioned SNQPs, the problem of handling unexpected node failure events was reduced to a routing problem between a node and the base station, due to either all nodes containing the same QEP or only the nodes that sense. This means there is no need to consider the reallocation of in-network operators that were placed upon the node that experienced the unexpected node failure event. In the SNEE SNQP, this is not the case, as nodes are installed with the minimum amount of code required to execute their allocated purpose, which results in the nodes being incapable of handling the load of any other node that has experienced such an event, unless the node is reprogrammed during the adaptation process. The SNEE SNQP (as described in [32]) has no adaptive behaviour to mitigate the effects of unpredictable node failure events.
3.5.1 Routing Based

Others in the WSN community takes the view that a WSN is a data collection system without any in-network processing [5, 88, 38, 28, 123, 25]. As a result, the problem of handling unpredictable node failure events has been tackled by a multitude of approaches which focus on re-routing data around the node that experienced the event, and so are of limited use when considered in conjunction with in-network operations. The proposals can be categorised into two distinct behaviours, namely multi-route and replication.

Multi-route proposals store alternative routes inside each node’s routing table, and when a node experiences an unpredictable node failure event the child redirects its tuples to its alternative parent [5, 88, 38, 28]. The drawback of this is that every node in the deployment needs to be active all the time, so that it is ready to receive data from any of the nodes that it becomes a parent to. There is also no consideration of the timing issue associated with this type of routing. Therefore, two or more nodes could start communicating with the same parent at the same time by both selecting the same alternative node.

Replication proposals aim at a best-effort approach by replicating the packet over several routes, in the belief that one version of the packet will eventually reach the base station [123, 25]. There are many drawbacks to this type of approach, of which the main ones are that it assumes that communication is cheap and that transmissions do not interfere with each other through collisions. These assumptions do not hold in deployments in which SNQP’s are targeted, where communication is the most energy-expensive operation a node can execute, and wireless signals can inadvertently interfere with each other resulting in the packets being lost.
3.5. RELATED WORK

3.5.2 Hybrid

There are proposals that execute limited in-network processing, e.g., by aggregating data as it travels downstream towards the base station. Such proposals have to handle the reallocation of aggregation operations as well as re-routing of data when faced with unpredictable node failure events. Many of these proposals are hierarchical in nature, as they break the deployment into a collection of two or more layers where most nodes in each layer only communicate with nodes in the same layer [56].

In proposals which break a deployment $d$ into two layers [56, 76, 126, 89, 10, 124, 71], the first layer consists of a collection of clusters, each of which contain a subset of the nodes in $d$. In each cluster, the nodes communicate with each other, and one node is selected to participate in the communication of the second layer, referred to as the cluster head. The second layer consists of the collection of cluster heads which communicate either directly to the base station or with each other in a multi-hop fashion. Aggregation operations are allocated to cluster heads, and when an unexpected node failure event occurs it can happen either to a cluster head or to an ordinary node in the cluster. The failure of the latter results in no action, but the failure of a cluster head initiates a new cluster head selection process.

Many of these proposals assume that the active cluster heads can always create a non-partitioned network, which, in the case of deployments that SNQPs target, is not always possible. There is also no consideration of the effect that the cost of determining a new cluster head has on the lifetime of the deployment. Finally, the cluster head selection process is often based on local information, which may not result in a decision that is suitable for the entire deployment.
3.5.3 Node Redundancy

The view of a WSN as a data collection system without any in-network processing means that parameters such as coverage (how many sensors are actively taking readings in a given geographical area) [17, 61] or degree of node redundancy (nodes that could be but are not currently active) [97, 122, 121] can be adjusted to support resilience [17, 61].

3.6 Conclusions

In this chapter, two different strategies for adapting to unpredictable node failure events have been presented. Their aims and designs have been described through the use of examples. The main conclusion from the evaluation is that both adaptive strategies result in significantly longer functional lifetimes compared to a non-adaptive strategy. Adaptive strategies, therefore, produce better BFB values than not adapting. However, due to the low-energy footprint of the adaptations, neither strategy produces significantly higher lifetimes over each other on a regular basis, and therefore choosing an adaptive strategy for maximum lifetime remains an open issue. In contrast, selecting a strategy that produces the minimum disruption to the continuity of the results stream is easier. This is because the regional re-optimisation strategy regularly produces significantly faster adaptations composed to the complete re-optimisation strategy and therefore is better able to support adaptations that are seamless from the end users point of view.

There is little related work in the SNQP literature on adapting to unpredictable node failure events. Options include switching parent nodes in TinyDB [80] or using routing protocols to ensure end-to-end tuple delivery as in AnduIN [66].

In Chapter 4, the problem of adapting to predictable node failure events is addressed by a strategy that prolongs the functional lifetime of the deployment
by reallocating in-network operations to different nodes over the lifetime of the deployment in such a way as to avoid uneven energy depletion across the deployment. In Chapter 5, a general framework for adapting to both types of unpredictable failure events (unpredictable node and unpredictable communication channel failure events) discussed in this thesis is presented. The general framework exploits planned redundancy to ensure a minimum level of resilience to both unpredictable node and unpredictable communication channel failure events.
Chapter 4

Resilience to Predictable Node Events

This chapter presents a technique designed to prolong the lifetime of a deployment given predictable node events, such as node failure due to energy depletion as a result of running a QEP generated by the SNEE SNQP described in Section 2.3. As mentioned previously, the SNEE SNQP follows the operational paradigm of compile/optimise once, execute many times for its QEPs, which are not adaptive. Therefore, no account is taken of the effect that fragmentation and distribution of workloads has on node energy levels over the lifetime of the QEP. In practice, the distribution of workloads results in different rates of energy drain at each participating node and therefore some nodes run out of energy earlier than others, thereby forcing the infrastructure to adapt or else cease QEP execution.

For example, consider two identical nodes $n$ and $n'$. At some $n$ that has to: (1) receive tuples from its children, (2) sense the environment using its onboard sensors, (3) execute in-network operations on the collection of tuples, and finally (4) transmits the resulting tuples onward. Firstly assume $n'$ that has to: (1) sense the environment using its on board sensors, and (2) transmit the obtained data
onward. Assuming all output tuples from $n'$ are input to $n$, then $n$ is guaranteed to run out of energy earlier than node $n'$ if both nodes started with the same initial energy stocks.

A SNEE-generated QEP is tightly controlled by a global agenda (as discussed in Section 2.3.4.3), and the workload allocated to a given site is specified by the process of fragmentation and distribution of operators (as discussed in Section 2.3.4.2). Moreover, both the where- and when- scheduling processes are grounded on empirically-validated cost estimation models [16], which include the ability to predict energy consumption. It is, therefore, possible to estimate the lifetime of each site in an ideal environment through the use of the SNEE energy consumption model. Assume a query $Q$ is compiled to execute over a deployment $D$ which costs $x$ to deploy. Now assume the compiled QEP $p$ is expected to produce $y$ tuples per agenda cycle. In this thesis, the estimated lifetime $t$, is the number of agenda cycles until the first site will fail due to energy depletion. If $p$ is non-adaptive (as is the case for the version of SNEE described in [32]). This would result in a BFB value of $\frac{y}{x}$. As shown in Chapter 3, it is possible to extend the lifetime of $Q$ by adapting to the node failure once it has occurred. However, as discussed in Chapter 3, reactive adaptation may limit the opportunities for further adaptations and therefore may fail to achieve higher BFB values because they do not consider the effect of completely draining a node of $p$.

The hypothesis explored in this chapter is that lifetimes can be extended by switching from a QEP $p$ to a different QEP $p'$ before $p$ drains any of its nodes, may give rise to a lifetime $t'$ of $p'$ that is greater than the lifetime $t$ of $p$. This results in a BFB’ value of $\frac{y}{x}$, where BFB’ $> \text{BFB}$ since $t' > t$ for the same $x$ and $y$.

The technique described in this chapter generates a sequence of timed transitions between functionally equivalent QEPs that extends the lifetime of query
CHAPTER 4. RESILIENCE TO PREDICTABLE NODE EVENTS

execution before any node fails due to energy drain. The different QEPs in a sequence result in tuples being routed to different sites at different points in time. As a result, such sites have different workloads allocated to them at different points in time. Based on Chapter 3: Run $q$ until it fails. Then try to construct a $q_i$, run it till it fails. In contrast with generating a plan sequence $[<q,t_i>,<q_{i+1},t_{i+1}>,<q_n,t_n>]$, such that $t_i \leq t_{i+1}$. Now assume that in $q$, the node that fails first is an acquisition node. The strategies discussed in Chapter 3 could not adapt to such a failure, whereas the strategy described in this chapter could shift the workload off the acquisition node before it is drained completely and therefore increase the lifetime of the node.

The contributions described in this and the previous chapter are designed to work on deployments where there is unplanned-for redundancy. The discussion on how to use planned redundancy to design deployments in such a way as to ensure a given degree of resilience to failure events is postponed until Chapter 5.

4.1 Overview

Using a sequence of plan transitions is mainly a proactive approach. In essence, a plan sequence is generated with the aim of exploiting the available resources whilst avoiding a complete drainage by transitioning proactively to another plan before an adaptation is necessary in response to a node failure. However, should an unpredictable node failure event cause node failure, thereby invalidating the sequence, a new sequence is computed that reacts to the new resource limitations. Thus, the strategy is mostly about proactive adaptations, but is capable of reactive adaptation as well. At compile time, an initial sequence is generated, which is followed until faced with an unpredictable node failure event. If an unpredictable node failure event occurs, the technique exploits the contributions described in
4.1. OVERVIEW

Chapter 3 to react to the node failure. It does so by generating a new QEP which seeds the process of computing a new sequence of plan transitions taking into account the new deployment structure. The infrastructure discussed in Section 3.2.3 is, therefore, extended as shown in Figure 4.1. We refer to the extended system as the proactive version of ADSNEE, referred to as ADPSNEE. The sequence generator is placed after the original SNEE compilation stack (Stage 1.1 in Figure 4.1). To handle QEP transitions, a sequence timer is introduced that tracks the number of completed agenda cycles and triggers the adaptation procedure described in Section 3.2.3. This is because the transition to the next QEP in the sequence to the deployment, resembles in all other respects the process of an adaptation, albeit a proactive, rather than reactive, reprogramming.

Figure 4.1: The ADPSNEE infrastructure.
4.2 The Sequence of Plan Transitions

A sequence of plan transitions, as described in this chapter, aims to contain QEPs that differ in terms of how tuples are routed through the network and where the in-network computation takes place. A sequence of plan transitions also considers when to change between these QEPs in such a way as to maximize the lifetime of the deployment by avoiding predictable node failure events, i.e., those stemming from energy depletion. A plan transition represents the process of changing between a QEP executing on $D$ and a functionally equivalent, yet different, QEP that is planned to start executing. The sequence generation process is depicted in Figure 4.2 and corresponds to Step 1.2 in Figure 4.1. The process, as a whole, can be thought of as dealing with three distinct sub-problems. These are:

1. How to generate a sequence of plan transitions containing functionally equivalent QEPs which result in long lifetimes. This is addressed through the use of tabu search [44] to explore alternative sequences (as described in Section 4.2.1, and represented by the Sequence Generator in Figure 4.2).

2. How to generate effective candidate plans within the plan sequences. This is addressed through the use of an evolutionary search to identify alternatives (as described in Section 4.2.2 and represented by the Neighbourhood Generator in Figure 4.2).

3. How to react to an unpredictable node failure event that disrupts the current sequence of proactive adaptations. This is addressed by exploiting a variant of the complete re-optimisation technique described in Chapter 3, which is used to generate a new initial QEP that enables a new sequence of plan transitions to be computed (as described in Section 4.2.3 and represented...
4.2. THE SEQUENCE OF PLAN TRANSITIONS

4.2.1 The Sequence Generator

The goal of the sequence generator is to produce an ordered collection of plan transitions between different, yet functionally equivalent, QEPs. Each transition is associated with a start time. A sequence of plan transitions is of the form 

\[[p_0, t_0], ..., (p_n, t_n)\],

such that each element consists of a QEP \(p_i\) and the start time \(t_i\) in agenda cycles at which \(p_i\) must replace \(p_{i-1}\) as the actively executing QEP. The start time of the initial plan in the sequence is \(t_0 = 0\).

The sequence generator uses tabu search [44] to explore alternative sequences of plan transitions and identify sequences that lead to the best lifetimes. Tabu search is often used to solve optimization problems where an optimized ordering and selection for a given goal is desirable [42, 43]. This, therefore, makes it

Figure 4.2: The strategy at a glance.

by the Run Time Module in Figure 4.2).
CurrentBestSolution(Solution initialSolution, Environment e)
1  Solution currentBestSolution ← initialSolution
2  Solution currentSolution ← initialSolution
3  List<Solution> tabuList ← []
4  tabuList.add(currentBestSolution)
5  while not STOPPINGCONDITION() do
6    List<Solution> candidateList ← []
7      for (candidate in NEIGHBOURHOOD(currentSolution, e) do
8        if NOTTABUED(candidate) or MEETSASPIRATION(candidate)
9          then candidateList ← candidateList + candidate
10         end if
11        end for
12      currentSolution ← LOCATEBESTCANDIDATE(CANDIDATELIST)
13      if FITNESS(currentSolution,e) > FITNESS(currentBestSolution,e)
14        then currentBestSolution ← currentSolution
15        end if
16      tabuList ← UPDATETABULIST(currentSolution,
17          currentBestSolution, tabuList)
18      if ! reached a local maximum then
19        currentSolution ← DIVERSITYMETRIC(currentSolution)
20      end if
21      tabuList ← REMOVEELEMENTSFROMTABULIST(
22          currentSolution, tabuList)
23  end while
24  return currentBestSolution

Figure 4.3: Computing the current best solution.

suitable for exploring the ordering of the transitions. A key feature of tabu is
that it actively avoids searching solutions that it has previously considered (unless
they meet an aspiration criterion) through the use of a tabu list \(^1\) that contains
previously considered solutions.

Figure 4.3 provides pseudocode for a generic tabu algorithm. The specific
instantiations of conditions, functions, etc., are described later in this section.
A tabu search starts with an initialSolution, which in this case is the sequence
\([(p_0,t_0)]\), where \(p_0\) is the plan generated by the SNEE compiler/optimizer for
the given query \(q\), and time \(t_0 = 0\), i.e., the plan is to be run from the start. Note
that it has no scheduled completion time as indicated by the fact that it has
no successor. In the running example, \(p_0\) is represented by the QEP shown in

\(^1\)The word “tabu” is a variant of “taboo”, meaning prohibited practice.
4.2. **THE SEQUENCE OF PLAN TRANSITIONS**

![Diagram](image)

**Figure 4.4:** An example of a sequence of QEPs.

Figure 4.4(a). The goal of the tabu search is to identify sequences of transitions with QEPs \((p_0, \ldots, p_n)\) and associated switch times \((t_0, \ldots, t_{n-1})\) that improve on the estimated functional lifetime of the deployment. Lines 1-4 both indicate that the \textit{initialSolution} is to be considered the best until a better one is found, and initialize the \textit{tabuList} in a way that discourages the revisiting of the \textit{initialSolution}. The \textit{Environment} parameter provides any additional information that may be needed to generate the neighbourhood or to assess the fitness of solutions, which in this case is the metadata about the query and the deployment, as described in Section 2.3.2.

In each iteration (lines 5-16), a neighbourhood is generated from which the best solution that either is not tabued or else meets an aspiration criterion is selected (Lines 5 - 10). Given a candidate solution, an \textit{aspiration criterion} is a success measure that overrules the presence of that candidate in the tabu list. In our case, the neighbourhood generator produces candidate plan solutions that extend the \textit{currentSolution}, using the evolutionary search algorithm described in Section 4.2.2, where the \textit{Seed} is the query and the \textit{Environment} is the metadata about the network required to allow the lifetime of candidate successors to be estimated using a cost model.
A fitness function is used to compare the currentSolution produced by the neighbourhood to the currentBestSolution. If the former represents an improvement compared with the currentBestSolution then it becomes the currentBestSolution (Lines 11 - 12 in Figure 4.3). In Line 13, the currentSolution is added to the tabu list, thereby preventing the search from considering this solution again (modulo the aspiration criteria).

If a position is reached where it is impossible to improve on the currentBestSolution (Lines 14-16), a diversity metric is applied with a view to escaping from a local maximum in the pursuit of a global maximum. A diversity metric may, for example, cause a return to a previous state or a jump to a completely new state at a random location in the search space. After each cycle, a STOPPINGCONDITION determines if the algorithm should return the currentBestSolution and exit.

The remainder of this section describes how the generic tabu algorithm in Figure 4.3 has been applied to QEP sequence transition generation by describing:

- The structure of the tabu list;
- The generation of neighbourhoods;
- The fitness function;
- The tabu list update function;
- The diversity metric;
- The stopping condition.

The example from Chapter 3 is used for consistency, and results in the sequence depicted in Figure 4.4. Running the algorithm on this topology gives rise to a step-by-step evolution of the tabu list, which is depicted in Figure 4.5.
### 4.2. THE SEQUENCE OF PLAN TRANSITIONS

<table>
<thead>
<tr>
<th>Pos</th>
<th>Plans Tabued</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>1</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>2</td>
<td>QEP₀, any</td>
</tr>
</tbody>
</table>

#### (a) Initial Stage.

<table>
<thead>
<tr>
<th>Pos</th>
<th>Plans Tabued</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>1</td>
<td>QEP₁, a</td>
</tr>
</tbody>
</table>

#### (b) After QEP 1 is added to the current tabu list.

<table>
<thead>
<tr>
<th>Pos</th>
<th>Plans Tabued</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>1</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>2</td>
<td>QEP₂, b</td>
</tr>
</tbody>
</table>

#### (c) After moving to a new position.

<table>
<thead>
<tr>
<th>Pos</th>
<th>Plans Tabued</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>1</td>
<td>QEP₁, a</td>
</tr>
</tbody>
</table>

#### (d) After QEP 2 was added to the sequence.

<table>
<thead>
<tr>
<th>Pos</th>
<th>Plans Tabued</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>1</td>
<td>QEP₁, any</td>
</tr>
<tr>
<td>2</td>
<td>QEP₂, b</td>
</tr>
</tbody>
</table>

#### (e) After diversity metric moved to position 1.

<table>
<thead>
<tr>
<th>Pos</th>
<th>Plans Tabued</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>1</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>2</td>
<td>QEP₂, c</td>
</tr>
</tbody>
</table>

#### (f) After diversity and QEP 2 selected at the new time c.

<table>
<thead>
<tr>
<th>Pos</th>
<th>Plans Tabued</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QEP₀, any</td>
</tr>
<tr>
<td>1</td>
<td>QEP₁, a</td>
</tr>
</tbody>
</table>

**Figure 4.5:** Step by step evolution of the tabu list.

#### 4.2.1.1 Tabu List Structure

The tabu list contains information about previously visited solutions with a view to avoiding repeated exploration of the same parts of the search space. In our case, each item in the tabu list is itself a list of the form \([(p₀, t₀), \ldots (pₙ, tₙ)]\), such that each \(p_i\) is a QEP, and each \(t_i\) is either:

- A specific start time of \(p_i\) in agenda cycles.

- The value *any*, which represents any start time that could be associated with \(p_i\).
An entry in the tabu list of the form \([\ldots(p_j, t_j)\ldots]\) indicates that the search should not further consider the transition with plan \(p_j\) with start time \(t_j\) as a candidate transition at position \(j + 1\), and an entry of the form \((p_j, \text{any})\) indicates that the search should not consider transitions with \(p_j\) with any start time at position \(j + 1\).

In the example, Line 3 of Figure 4.3 initializes the tabu list, and Line 4 adds the initial QEP \((QEP_0)\), represented in Figure 4.4(a), into the first position in the list with an \(\text{any}\) start time (that the start time is \(\text{any}\) is implemented within the \(\text{add}()\) operation); the updated tabu list is as shown in Figure 4.5(a). This restricts the search so that the initial QEP is unlikely to be revisited later in the search.

As mentioned earlier, a tabu list can have an aspiration criterion. In our case, the aspiration criterion is that a tabued solution must give at least a 5% increase in predicted plan lifetime compared with the current best solution for it to be considered. This effectively allows a cycle in the transitions in a sequence, but only if a significant improvement can be obtained by doing so.

### 4.2.1.2 Generating the Neighbourhood

The key functional components of the tabu search are the exploration of the neighbourhood and the maintenance of the tabu list. The neighbourhood is a collection of candidate solutions. In our case, a candidate solution is a plan sequence that consists of the \(\text{currentSolution}\), extended with an additional transition identified by the genetic search described in Section 4.2.2. The \(\text{NEIGHBOURHOOD}\) function itself is defined in Figure 4.6, and is essentially a wrapper that adds sequence transitions returned by an evolutionary search to the \(\text{currentSolution}\). This corresponds to Phase 1.1 in Figure 4.2.
4.2. THE SEQUENCE OF PLAN TRANSITIONS

\[ \text{NEIGHBOURHOOD}(\text{Seed } s, \text{ Environment } e) \]

1. \( \triangleright \) Adds each of the elements in the elite list
2. \( \triangleright \) to the plan sequence that was provided as the Seed \( s \)
3. List\( <\text{Solution}> \) neighbourList = [];
4. int pos = 0;
5. for (c in GENETICSEARCH(Seed s, Environment e)) do
6.   neighbourList[pos] = new Solution(s, c);
7.   pos++
8. return neighbourList

Figure 4.6: Computing the neighbourhood.

4.2.1.3 Fitness Function

The effectiveness of each currentSolution from the neighbourhood is compared to that of the currentBestSolution (in Line 11 of Figure 4.3, corresponding to Phase 1.2 in Figure 4.2) using a fitness function. The fitness function estimates the overall lifetime of a sequence of plan transitions, taking into account the costs of transitioning between each \( p_i \) and \( p_{i+1} \). Figure 4.7 describes the fitness function, where \( N \) denotes the set of nodes in the network, \( S \) denotes the plan sequence, and \( QEP\text{Cost}(n, p) \) represents the cost of running the part of QEP \( p \) that is assigned to site \( n \in N \) for one agenda cycle.

In Figure 4.7, Formula (4.1) defines the fitness to be the lifetime of the shortest lived site contributing to the last plan in the sequence. The lifetime of a site \( n \) in agenda cycles is estimated from the energy levels left when the final transition has completed and the cost of executing an agenda cycle of the final QEP on \( n \). Formula (4.2) estimates the remaining energy on site \( n \) by subtracting from the original energy stocks of the node the initial programming cost, the total costs of running the previous QEPs, and the adaptation cost of transitioning between previous QEPs. Formula (4.3) determines the total running cost of a QEP \( q \) for a node \( n \) given the period in agenda cycles for which the QEP is planned to run.
fitness(S) = \min \left\{ \frac{\text{leftOverEnergy}(n, S)}{\text{QEPCost}(n, \text{last}(S))} \mid n \in N \right\} \tag{4.1}

\text{leftOverEnergy}(n, S) = n.\text{initialEnergy} - (\text{initialProgrammingCost}(n, q_0) + \sum_{i \in 0 \ldots (|S| - 1)} (\text{runningCost}(n, q_i) + \text{adaptationCost}(n, q_i, q_{i-1}))) \tag{4.2}

\text{runningCost}(n, q) = \text{QEPCost}(n, q) \ast q.\text{switchTime} \tag{4.3}

Figure 4.7: The fitness function.

To estimate the adaptation cost from one plan to another the same adaptation breakdown procedure and estimates as described in Section 3.2.3 are used. In the example, the fitness of the initial QEP ([\text{QEP}_0, 0]), corresponding to the currentBestSolution, is compared with the fitness of the currentSolution \([\text{QEP}_0, 0], (\text{QEP}_1, a)]\), where \text{QEP}_0\ is run for \(a\) agenda cycles and then switched to \text{QEP}_1.

### 4.2.1.4 Tabu List Update Function

The length of the tabu list tracks that of the currentSolution. In essence, whenever the current sequence of plan transitions, represented by the currentSolution, is extended, the tabu list is extended to capture the area of the search space that has been explored during the production of the currentSolution. In addition, whenever the current sequence of plan transitions is shortened by the application of a diversity metric (described below) the tabu list is shortened by a corresponding amount.

Once the currentBestSolution has been determined for each iteration, the tabu list is updated (Line 13 in Figure 4.3) to reflect the decision made (in Line 11). This corresponds to Phase 1.3 in Figure 4.2.
4.2. THE SEQUENCE OF PLAN TRANSITIONS

If the currentSolution improves on the currentBestSolution, then, assuming that the tabu list is of length $l$, the tabu list gains the following entries:

1. the last transition in the currentBestSolution sequence of plan transitions is added to the list at position $l$ in the tabu list; and
2. the tabu list is extended to length $l + 1$, at which a list is created that contains an entry for every transition in the currentBestSolution with a time template of any, thereby steering the search away from QEPs that have given rise to the currentBestSolution.

In the example, if the sequence of plan transitions $[(QEP_0, 0), (QEP_1, a)]$ returns a better estimated lifetime than running $QEP_0$ until a node fails, then the transition $(QEP_1, a)$ is added to the list at the last position in the tabu list, and then the tabu list is extended with a new element that contains $[(QEP_0, any), (QEP_1, any)]$; this results in the tabu list represented in Figure 4.5(c).

If the currentSolution does not have better fitness that the currentBestSolution, then, assuming that the tabu list is of length $l$, the last transition in the sequence in the currentBestSolution is added to the list at position $l$ in the tabu list. In the example, if the sequence of plan transitions $[(QEP_0, 0), (QEP_1, a)]$ had returned a worse estimated lifetime, then the only addition to the tabu list would have been the transition $(QEP_1, a)$, resulting in the tabu list represented in Figure 4.5(b).

4.2.1.5 The Tabu Diversity Metric

If the currentBestSolution has the better fitness (Line 11 in Figure 4.3), then the search may be at a local maximum from which it needs to escape if a global maximum is to be found. A local maximum is characterized by some threshold number of iterations that have led to no improvement to the currentBestSolution.
It was determined experimentally that, after three iterations without improvement, the strategy would rarely improve upon the solution. Therefore, we use three iterations without improvement to define a local maximum. This behaviour is captured in Line 15 in Figure 4.3, and represents the first stage of Phase 1.4 in Figure 4.2.

With the goal of escaping from local maxima, a diversity metric is used that randomly chooses a transition in the currentBestSolution that it had previously visited as a starting point for future iterations. In the example, after the search has failed 3 times to locate a currentSolution with better fitness than the currentBestSolution, which is \([(QEP_0,0),(QEP_1,a),(QEP_2,b)]\), the search is assumed to be in a local maximum. The diversity metric chooses (at random) to move back 1 transition, and thus the sequence \([(QEP_0,0),(QEP_1,a)]\) is used as the starting point for continuing the search.

When the diversity metric is applied, the tabu list is updated so that its length is consistent with that of the currentSolution. This allows previously visited plans to contribute to new sequences. This behaviour is captured in Line 16 in Figure 4.3, and represents the second stage of Phase 1.4 in Figure 4.2. In the example, at this point, the tabu search would remove the last entry from the tabu list, yielding the result in Figure 4.5(e). This allows the tabu search to reconsider the transition QEP_2 at times other than b. Therefore, if the search were to carry on and choose QEP_2 at time c, and the sequence of plan transitions \([(QEP_0,0),(QEP_1,a),(QEP_2,c)]\) were to give a better lifetime than the currentBestSolution consisting of \([(QEP_0,0),(QEP_1,a),(QEP_2,b)]\), then the resulting tabu list would be as shown in Figure 4.5(f).
4.2.1.6 Stopping Criterion

When each iteration finishes, if any of the following conditions are satisfied, the algorithm finishes and returns the currentBestSolution. The constants in the conditions were identified empirically, through experiments on a sample of three topologies, where different iteration values were used and compared. When iteration values above 200 were used, the end sequence generated little or no difference in terms of expected lifetime. Furthermore, once the system had settled on a solution for more iterations than the length of a sequence, it would rarely find a better solution during the rest of the iterations. Therefore the empirically-established stopping criterion is either or both of:

- 200 iterations have taken place; or

- the algorithm has chosen the same currentSolution for \( n \) iterations, where \( n \) is the largest of either 2 or the maximum length of a sequence during the search.

This corresponds to Phase 1.5 in Figure 4.2.

4.2.2 Neighbourhood Generation

The neighbourhood for the tabu search identifies candidate transitions whilst taking into account the energy left over by the previous QEPs in a sequence. Generating diverse QEPs, that are equivalent to the original QEP but different in terms of their patterns of energy usage on the nodes, is a challenging problem, due to the large search space from which these QEPs can be generated. An evolutionary approach is used to generate the neighbourhood for the following reasons:
• It can provide a collection of transitions (solutions) that qualifies as a neighbourhood for the Tabu search.

• It is often capable of finding good solutions efficiently \[87\].

• An appropriate proposal for generating routing trees existed in the literature \[64\], upon which we could build.

Evolutionary search is inspired by biological evolution, in that it starts with a population of individuals that is then evolved. In our case, each individual represents a candidate transition, encoded as a binary string. Such a representation is referred to as a genotype and is mappable to a corresponding phenotype, which represents application concepts more explicitly. At each generation, a fitness function determines pairs of individuals that are chosen to reproduce (i.e., be the basis for the derivation of offspring) and therefore contribute to the next generation \[46\]. The pseudocode for a generic evolutionary algorithm is shown in Figure 4.8. During the reproduction phase, the genotypes of each parent can both be mutated through random changes and merged by a crossover function.

The remainder of this section indicates how the generic algorithm in Figure 4.8 has been applied to the generation of a collection of query plans and switch times, by describing the genotype representation used, the generation of an initial population of genomes, the mutation and crossover operators, and the stopping condition.

4.2.2.1 Genotype Representation

Our genetic search uses a genotype that represents the nodes that can participate in query evaluation and a switch time. The genotype is a vector of binary values, in which each of \( n \) bits represents one node in the network, and \( m \) bits encode an integer representing the switch time.
4.2. THE SEQUENCE OF PLAN TRANSITIONS

ELITELIST(Seed s, Environment e)
1 Genome master = CREATEMASTERGENOME(s)
2 List<Genome> eliteList ← []
3 List<Genome> population ← GENERATEINITIALPOPULATION(s, e)
4 while not STOPPINGCONDITION() do
5 List<Genome> newPopulation ← []
6 population ← ORDEREDBYFITNESS(population, e)
7 eliteList ← GROWELITELIST(eliteList, population)
8 while newPopulation.size() < population.size() do
9    Iterator i = iterator over the population list
10   Genome p1 = i.next(), Genome p2 = i.next()
11   with probability p: mutate p1, p2
12      ▶ c1 and c2 ← CROSSOVER(p1,p2)
13      ▶ c1 and c2 ← OR(master, c1, c2)
14   population ← newPopulation ∪ { c1, c2}
15 return eliteList

Figure 4.8: Computing the elite list.

For the $n$ bits that represent node participation, a 1 indicates that the node is to be active, and a 0 that it is not. The query optimizer is then constrained to generate a QEP that only uses the active nodes.

The $m$ bits that represent the switch time encode the predicted lifetime of the current query plan in terms of agenda cycles. This is represented in Figure 4.9.

In addition to the genome, a master genome is used (inspired by [64]), that has 1s for the nodes that must be active in the query (i.e., the nodes that sense data and the sink node). In this way, taking the result of bitwise OR operation between a genome with the master genome one can ensure that every candidate genome has at least the acquisition nodes and sink node set as active, which is a correctness condition on every QEP.
4.2.2.2 Generating the Initial Population

The \texttt{generateInitialPopulation} function, (Line 3 in Figure 4.8) takes as input the query being evaluated as the \textit{Seed $s$}, and the information about the deployment described in Section 2.3.2 as the \textit{Environment $e$}. It starts by generating a set of QEPs that initialise the search with a set of working QEPs. These QEPs are generated by the query optimizer on the basis of a routing tree (RT) generation algorithm that can use different heuristics to generate diverse RTs [31]. These RTs are then used to underpin the generation of individuals. If more individuals are required to reach the desired population size, once the seeds are created, additional individuals are generated randomly, and or\textsuperscript{ed} with the master genome for correctness. This represents Phase 2.1 in Figure 4.2.

The switch times are chosen at random between $0$ and the estimated lifetime of the previous QEP. The reasoning for using a random selection for the switch times is that, due to the decisions made on previous transitions, it would be very difficult to select, for each individual at every point in time, the best switch time.

4.2.2.3 Mutation and Crossover

The population for the next generation (\textit{newPopulation}) is created from the previous generation (\textit{population}) by applying a fitness function to select the individuals that will become parents by combination of their genomes. The fitness of the individuals is estimated using a cost model that relates the predicted cost on each node to the energy stocks on the node. Individuals that do not generate a valid QEP are given a fitness of zero. The population is then used to grow the \textit{eliteList}, which is the collection of the best $n$ genomes produced so far, where $n$ is the population size (30 was determined empirically to be an appropriate population size in
terms of speed of execution and result quality). Individuals with the highest fitness are combined (i.e., become parents and generate offspring) to preserve their beneficial properties. Some individuals with low fitness also generate offspring in order to encourage diversity in the population. It is worth noting that the SNEE optimizer is considered a black box component by the genetic algorithm, therefore, assumptions made by the optimiser are reflected in the QEPs produced by the genetic search algorithm.

The mutation operator, which is run with a probability of 0.5 on each parent can do either or both of:

- Flip a random bit in the section of the genome that represents the active nodes in the topology;

- Use Gaussian convolution [78] to find a value between 0 and the estimated lifetime of the previous QEP for the switch time.

The crossover operator combines random portions of the genomes of the two parents using uniform crossover [78], as illustrated in Figure 4.10. After the two new genomes are made, they are or’ed with the master genome, to produce two children genomes for the next generation, that at least contain the acquisition nodes and sink node. This corresponds to Phase 2.2 in the system represented in Figure 4.2. The reasoning for using each pair of parents, including the low fitness ones, is to include more diversity in the next population.

4.2.2.4 Stopping Criteria

The genetic algorithm exits when 50 generations have been investigated, or there has been no improvement for 5 generations (these constants were also determined empirically). If the termination condition is met, the current generation
4.2.3 Run Time Module

The role of the runtime module is to support the strategy based on plan transitions when unpredictable node failure events occur. Unpredictable node failure events may cause a sequence of plan transitions $S$ to become invalid, as the running QEP, and/or its successors, may depend upon a node that is no longer available. To prolong the lifetime, a new sequence of plan transitions must be produced that takes into account the unpredictable node failure event.

To generate a new sequence of plan transitions, the connectivity graph is updated and used by the SNQP to account for the unpredictable node failure event by producing a new Environment variable $e'$, corresponding to Phase 3.1 in Figure 4.2. The optimizer then generates a new QEP $p'_0$ given $e'$ that becomes the seed for generating a new sequence from this point forward, corresponding to Phase 3.2 in Figure 4.2.
4.3. EVALUATION

155

4.3 Evaluation

The experiments reported in this section investigate whether the use of a derived sequence of plan transitions can increase the estimated lifetime of a query \( q \), on a deployment \( d \), thereby improving its BFB value. In doing so, the same assumptions as made as in the unpredictable node failure experiments, described in Section 3.4.1. The cost model to predict the lifetime of a sequence is the same as used in the unpredictable node failure experiments reported in Chapter 3.
The experiments are divided into four separate investigations:

- The first investigation (described in Section 4.3.2.1) aims to determine the extent to which sequences of plan transitions lead to increased query life-times, using a wide range of generated deployments and queries.

- The second investigation (described in Section 4.3.2.2) aims to determine how *unpredictable node failure events* effect the *sequence of plan transitions* in terms of lifetime.

- The third investigation (described in Section 4.3.2.4) aims to determine the time taken by adaptations that require switching between QEPs in a sequence.

- The forth investigation (described in Section 4.3.2.5) aims to determine, for individual sequences of plan transitions, whether the switch times selected in the transitions are appropriate for maximizing the lifetime of the deployment, given the QEP’s in the *sequence of plan transitions*.

### 4.3.1 Experimental Design

A set of 60 topologies were generated algorithmically, with corresponding physical and logical schemas. The set of topologies was subdivided into a 30-node subset a scale that is representative of typical environmental monitoring applications [23] and is referred to henceforth as the *small topology set*. The other subset contained 100 nodes, a scale that is representative of a large application [23] and is referred to henceforth as the *large topology set*.

Each query was compiled and executed with QoS Sets 1 and 2 from the QoS expectation sets defined in Section 3.4. Set 3 is not used in this experimental set up because, as discussed in Section 3.4, deployments with this QoS set are
dominated by the (impossible to avoid) costs of sleeping and sensing. These costs thereby make detecting changes that result from different techniques difficult to isolate.

For both topology sets, a third of the topologies were selected to be the execution platform for SELECT queries; a second third of the topologies were selected to be the execution platform for aggregation queries; and the last third of the topologies were selected to be the execution platform for a window-based join queries. The same partitioning procedure was applied on the large topology set.

Figure 4.12: A graphical representation of the sniper localisation deployment in the context of SNQP [103].

The deployment from the sniper localisation case study [103], as shown in Figure 4.12, is used to verify that the results presented here hold for a real life
CHAPTER 4. RESILIENCE TO PREDICTABLE NODE EVENTS

deployment. Because this case study is an event detection problem, some changes were made to bring it in line with a query processing context:

- A subset of the nodes was selected to sense the environment (whereas in the original all nodes were used for sensing) and the rest of the nodes were assumed to be available for relaying and/or in-network processing.

- Secondly, the set of sensing nodes was separated into two extents, at the left and right hand sides of the street in Figure 4.12, represented by the small squares and small circles, respectively, so as to facilitate the writing of a window-based join query.

- Thirdly, connectivity between nodes was limited to nodes that have a direct line of sight (no obstacles) with each other.

- Finally, as the original deployment was designed for event detection, each type of query (select, aggregation, and window based join) is executed with the fastest acquisition rate and delivery time achievable by SNEE.

Throughout the experimental results described in Section 4.3.2, the sequence of plan transitions, referred to as the proactive strategy, is compared with the complete re-optimisation strategy described in Chapter 3, which is referred to as the reactive strategy. For completeness, the sequence of plan transitions is also compared to the results obtained by original SNEE, which is referred to as the static strategy i.e., one that is not adaptive.

The decision to compare the sequence of plan transitions to the complete re-optimisation strategy instead of the regional strategy was made because the sequence of plan transitions uses the complete re-optimisation strategy within its runtime module, and therefore any difference in lifetime between them stems from the sequence of plan transitions.
4.3. EVALUATION

4.3.2 Results

This section describes the results obtained for each previously described investigation of the benefits stemming from sequences of plan transitions.

4.3.2.1 Stable Deployments: Lifetime Measurements

To determine the lifetime of a sequence of plan transitions $S$, $S$ must first be generated by inputting the topology, the corresponding query, and physical and logical schemas into the ADPSNEE compilation stack, and then taking the resulting QEP as input to the sequence generator, as shown in Stages 1.1 and 1.2 in Figure 4.2. Once $S$ has been generated, the energy cost model is used to estimate the energy levels left in each participating site at the point in time when the last transition is completed. These energy levels are then used, in conjunction with the cost of executing the QEP in the last transition for each site, to estimate the time before the first node fails from energy depletion. This is then used as the lifetime estimate. The logical flow diagram for this experiment can be found in Appendix G.3.1.

Figure 4.13 shows the predicted lifetime for the three strategies using QoS set 1 in Table 3.1, on the small topology set. The average performance improvements between the static and the proactive strategies for the different query types are shown in Table 4.1. The following can be observed:

- The proactive strategy yields increased lifetimes compared with static queries in all 15 cases. This is due to the use of nodes within the network that have relatively fresher batteries and due to the redistribution of workload over shared nodes between the QEPs. The proactive strategy results in an average improvement in lifetime of 29% compared with the static strategy.

- The proactive strategy yields increased lifetimes compared with reactive
CHAPTER 4. RESILIENCE TO PREDICTABLE NODE EVENTS

<table>
<thead>
<tr>
<th>Type of query</th>
<th>Average improvement over static</th>
<th>Average improvement over reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>26 %</td>
<td>19 %</td>
</tr>
<tr>
<td>Aggregation</td>
<td>8 %</td>
<td>8 %</td>
</tr>
<tr>
<td>Join</td>
<td>46 %</td>
<td>12 %</td>
</tr>
</tbody>
</table>

Table 4.1: Average improvement between proactive and the other two strategies for the different query types for QoS set 1 on the small topology set.

strategy in 14 of the 15 cases. This is mainly due to the first node that fails in the QEP being managed by the reactive strategy being an acquisition node, which a re-optimisation strategy is not able to adjust. In the proactive case, the adaptations extend the lifetimes of such nodes by adapting before they are drained of energy (for example, by moving operators off the node or by changing the nodes through which sensed tuples are routed).

The proactive strategy results in an average improvement in lifetime of 20 % compared with the reactive strategy.

- The reactive strategy performs better than the proactive strategy in one of the 30 cases (T15). In this case, both strategies improve significantly over the static strategy, and the increase in lifetime obtained from the reactive strategy is only 1% compared with the proactive strategy. This is explained by the non-exhaustive nature of the search approach used. The proactive strategy explores fruitful areas of the solution space but these are nevertheless different from and not as good as those the reactive strategy pursues.

The improvements for aggregate queries are lower than for select queries because the aggregate queries tend to transmit less data, and thus to have longer lifetimes. In such cases, the opportunities for adaptations that aim to extend lifetimes are reduced because of:

- the (impossible to avoid) cost of sensing, which constitutes an increasing fraction of the overall energy usage;
4.3. EVALUATION

<table>
<thead>
<tr>
<th>Type of query</th>
<th>Average improvement over static</th>
<th>Average improvement over reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>9 %</td>
<td>4 %</td>
</tr>
<tr>
<td>Aggregation</td>
<td>7 %</td>
<td>4 %</td>
</tr>
<tr>
<td>Join</td>
<td>9 %</td>
<td>6 %</td>
</tr>
</tbody>
</table>

Table 4.2: Average improvement between proactive and the other strategies for the different query types for QoS set 2 on the small topology set.

- the (impossible to avoid) cost of the network in sleep mode, which becomes increasingly significant.

By contrast, the improvements for join queries are higher than for select queries for the converse reasons. Join queries tend to have shorter lifetimes because the join operators give rise to relatively large results which need to be transmitted, therefore reducing the overall lifetime of the deployment. This dilutes the effect of the unavoidable sensing and sleeping times, and as a result the opportunities for effective adaptations increase.

Figure 4.14 shows the predicted lifetime for the three strategies described above using the QoS set 2 in Table 3.1 on the small topology set. This shows the effect associated with different QoS expectations. The average performance improvements between the static and proactive strategies for the different query types are shown in Table 4.2. The following can be observed:
CHAPTER 4. RESILIENCE TO PREDICTABLE NODE EVENTS

Figure 4.13: Lifetimes for static, proactive, and reactive strategies for QoS set 1 for the small topology set.

Figure 4.14: Lifetimes for static, proactive, and reactive strategies for QoS set 2 for the small topology set.
4.3. EVALUATION

The adaptive strategies both provide smaller gains than what was seen in QoS set 1. This is because the execution cost in QoS set 2 (where nodes sense much less frequently than in QoS set 1) is dominated by idle/sleep cost. As all nodes experience this cost, as explained in Section 3.4, there is less scope for lifetime extensions.

Both adaptive strategies still lead to lifetime extensions, showing that even in the face of increased idle/sleep cost both strategies perform better than the static approach.

The proactive strategy still produces an increase in lifetime compared to the reactive strategy.

Figure 4.15 shows the predicted lifetime for the three strategies described above using the QoS set 1 in Table 3.1 on the large topology set. The average
Table 4.3: Average improvement between proactive and the other strategies for the different query types for QoS set 1 on the large topology set.

<table>
<thead>
<tr>
<th>Type of query</th>
<th>Average improvement over static</th>
<th>Average improvement over reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>16 %</td>
<td>16 %</td>
</tr>
<tr>
<td>Aggregation</td>
<td>5 %</td>
<td>5 %</td>
</tr>
<tr>
<td>Join</td>
<td>27 %</td>
<td>14 %</td>
</tr>
</tbody>
</table>

The following can be observed:

- As with the small topology set, both adaptive strategies generate a significant lifetime extension in comparison to the static strategy.
- The proactive strategy, in all cases, outperforms the reactive strategy for the same reasons as discussed previously.

Figure 4.16: Lifetimes for static, proactive, and reactive strategies for the sniper deployment.
4.3. EVALUATION

<table>
<thead>
<tr>
<th>Type of query</th>
<th>Average improvement over static</th>
<th>Average improvement over reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>32%</td>
<td>0.2%</td>
</tr>
<tr>
<td>Aggregation</td>
<td>18%</td>
<td>18%</td>
</tr>
<tr>
<td>Join</td>
<td>32%</td>
<td>10.3%</td>
</tr>
</tbody>
</table>

Table 4.4: Improvement between proactive and the other strategies for the different query types on the sniper deployment.

Figure 4.16 shows the predicted lifetime for the three query types that were run upon the sniper deployment. The results show that even when maximizing the sensing and delivery rates (which reduces the opportunities for the optimizer to generate diverse plans that meet the QoS expectations), the proactive strategy produces an increase in the estimated lifetime of the deployment. Table 4.4 shows the percentage differences between the three strategies.

In conclusion, for deployments that are stable (in the sense that nodes only fail from energy depletion), planning adaptations proactively can increase the lifetime above what can be achieved when just reacting to node failure. By extending the lifetime of the query, the BFB value is increased, thereby making SNQP deployments more cost effective.

4.3.2.2 Unstable Deployments Lifetime Measurements

To generate lifetime estimates for sequences of planned transitions that may be disrupted by unpredictable node failure events, the same evaluation procedure as described in Section 3.4 is used, i.e., random nodes are selected to fail uniformly over the expected lifetime of the QEP. The same small and large topology sets as in Section 4.3.2.1 are used, with QoS set 1, and the different strategies are then compared. We do not consider QoS sets 2 or 3, because the dominant idle / sleep costs dilute the results. The flow diagram for this experiment can be found in Appendix G.3.2.
Figure 4.17 shows the estimated lifetime for the large topology set in the face of unpredictable node failures. The results show that most of the results lie above the diagonal (where the proactive strategy yields better lifetimes than the reactive strategy). There are three distinct clusters (which are denoted by A, B and C) in Figure 4.17. Each cluster results from a different type of query (select, aggregation and window-based join). The following can be observed:

- Cluster A represents the performance of select queries. In this case the proactive strategy outperforms the reactive strategy in all cases, and results in average increases in lifetime of 28%.

- Clusters B and C represent the performance from join and aggregation queries, respectively. The proactive strategy shows an increase in lifetime over the reactive strategy in 80% of the results, yielding average increases in lifetime of 69% and 6% for join and aggregation queries, respectively. In both cases where the reactive strategy outperforms the proactive strategy, the reactive strategy has been forced to explore a different yet fruitful search space from the proactive strategy: at each unexpected failure point, each strategy has had a different collection of nodes available for future use.

Figure 4.18 shows the estimated lifetime for the small topology set in the face of unpredictable node failures. The results show the same behaviour as the large deployment, though overall fewer nodes can fail before both strategies are unable to adapt to the failure (due to the lower level of available nodes for replacement).
4.3. EVALUATION

Figure 4.17: Lifetimes for proactive and reactive strategies in the face of unpredictable node failure for the large topology set.

Figure 4.18: Lifetimes for proactive and reactive strategies in the face of unpredictable node failure for the small topology set.
Table 4.5: Average and standard deviation for each query type.

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Average lifetime</th>
<th>Standard Deviation in lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>star</td>
<td>64446</td>
<td>4408</td>
</tr>
<tr>
<td>Aggregation</td>
<td>69707</td>
<td>451</td>
</tr>
<tr>
<td>Join</td>
<td>48443</td>
<td>1232</td>
</tr>
</tbody>
</table>

The overall conclusion is that, faced with unpredictable node failure events, the proactive approach outperforms the reactive approach on a regular basis, thereby leading to increased BFB values for deployments and supports the aim of making SNQP deployments more cost effective.

4.3.2.3 Robustness Measurements

In the lifetime measurement sections (see Sections 4.3.2.1 and 4.3.2.2) the same query (per type of query) was run over each topology (per topology set). To show that diverse queries of the same type do not have a significant effect on the results reported, we have taken a topology for which performance was average for each query type. Table 4.5 shows the average and standard deviation of each query type. The selected topology had randomly generated queries (generated by a random query generator) of that type executed over it. Their lifetimes were measured for both the adaptive and non-adaptive strategies. Figure 4.19 presents the different lifetimes produced by the various queries, and Figure 4.20 presents the difference between the strategies over different queries. The following can be concluded:

- The effect on the overall lifetime of a deployment can be significantly different (between 1.36 % and 59.65 % of the average lifetime) depending on the query. This is mainly due to the number of packets that have to be transmitted through the deployment as determined by the size of the time window as well as the size of each tuple (i.e., the number of attributes each tuple contains).
4.3. EVALUATION

- The overall effect of different queries of the same query type on performance between the adaptive and non-adaptive strategies is small, for the case of aggregation and join queries. The larger variance in the select queries is due to the different number of attributes being requested from each query. The difference from the average is 59.65 % for select queries, and 1.36 % for aggregation and 17.43 % for join based queries.
CHAPTER 4. RESILIENCE TO PREDICTABLE NODE EVENTS

Figure 4.19: The lifetime estimated for a given deployment over a collection of diverse queries.

Figure 4.20: The difference in lifetime between the different strategies over a collection of diverse queries.
4.3. **EVALUATION**

4.3.2.4 **Adaptation Time Measurements**

This section investigates the amount of time taken switching between QEPs in a sequence. Transitioning between each pair of QEPs in a sequence makes use of the collection of adaptation messages defined in Section 3.2.3. The time taken for these adaptations to be transmitted and processed is estimated using the time estimation models defined in Section 3.4.2.4.

Figure 4.21 shows the times taken by each switch in a sequence for the small topology set. The following can be observed:

- Adaptations between QEPs, which can be significantly different from each other, often requires the reprogramming of a significant number of nodes, and therefore often takes from 15 to 80 minutes to execute. This results in
a gap ranging 1 to 8 agenda cycles for QoS set 2.

- These adaptation times are similar to those incurred by the reactive strategy, and therefore there is little to choose between the strategies if the main goal is to maintain the continuity of the streamed results.

- It is often the case that the proactive strategy may require several adaptations during the lifetime of the deployment, resulting in a larger accumulated downtime than the reactive strategy. However, as shown in Section 4.3.2.1, the lifetime is larger than can for the reactive strategy.

For the agenda shown in Figure 2.15, to maintain continuity in the streamed results, any adaptation would need to complete within the 296 seconds, the golden zone of the agenda. In this case, none of the adaptations shown in Figure 4.21 would complete within this time. Increasing the golden zone so that it can support adaptations such as those seen in Figure 4.21 is possible by reducing the acquisition rate. In the case of the adaptations in Figure 4.21, acquiring data every 40 minutes would be enough support the average execution time of an adaptation. To support all the adaptations seen in Figure 4.21, the acquisition rate would need to be set at every 1.5 hours.

The main conclusion from this experiment is that the proactive strategy does indeed have a longer overall downtime over several adaptations (often of a magnitude of between three and seven times longer). For end users who need to have complete results while having resilience to both types of failures, then it is prudent to reduce the acquisition rate to levels where the golden zone can support adaptations and still return the results to the end user in time. If this is not possible, then the end user should not use either strategy.
4.3. EVALUATION

<table>
<thead>
<tr>
<th>generated sequence</th>
<th>optimal sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch 1</td>
<td>30548</td>
</tr>
<tr>
<td>switch 2</td>
<td>42282</td>
</tr>
<tr>
<td>lifetime</td>
<td>75392</td>
</tr>
</tbody>
</table>

Table 4.6: Lifetimes given different switch times between successive QEPs.

4.3.2.5 Switch Time Suitability

This section investigates how effective the switch times used in the transitions of a sequence of plan transitions are at maximising the lifetime that could be obtained for the QEPs within the sequence. The sequence of plan transitions generated for task $T_2$ in Figure 4.13 was selected, and the switch times in each transition were adjusted up and down by steps of 500 agenda cycles (a sample large enough to allow the experiment to be completed within a reasonable time period, yet not so large as to make it likely to miss any significant lifetime improvements). After each adjustment, the lifetime of the sequence is calculated and compared. The experimental flow diagram for this experiment can be found in Appendix G.3.3.

Figure 4.6 shows the results, in terms of estimated lifetimes that result from a pair of switch times. The best switch times identified yielded an increase in lifetime of an average of 1.5% in comparison to those in the original sequence (computed from Figure 4.6). Therefore, the adaptation times generated by the sequence were close to the best.

Figure 4.22 shows a comparison of the estimated lifetime of all sequences and the best possible switch times for the sequences generated for the large topologies. The following observations can be made:

- There is usually little difference (the greatest is 4.4%) between the original sequence and the optimal sequence; therefore the original sequence switch times were indeed suitably chosen. The average difference between the
original and optimal lifetimes (computed from all topologies that could be exhaustively searched in a time frame that allowed all the experiments to be run with the available resources) over all the sequences of plan transitions is approximately 0.43 %.

- With sequences that have many transitions, the time taken for the exploration of the different switch times increases near exponentially and therefore is omitted from the results. This results in topologies 2, 3, 4, 6, 13 and 15 not containing a result.

Figure 4.23 shows the sequence of plan transitions with the switch times and the expected lifetime of each QEP in the sequence. For each query, each bar in the chart represents a QEP in the sequence; the leftmost bar represents the initial
4.3. EVALUATION

Figure 4.23: The expected lifetime of the QEPs in each sequence generated from the small topology set with corresponding switch times.

QEP. For successors to the initial QEP, the unshaded portion of the bar is the interval before the switch time, the black portion is the interval for which the plan was predicted to execute, and the shaded portion is the predicted lifetime of the QEP. For example, for topology $T_2$, this shows that there are 2 adaptations, the first after 2,586 agenda cycles and the second after 31,250 agenda cycles. The following can be observed:

- Each QEP in a sequence gives an increase in lifetime over its predecessor QEP.

- The first transition produces significantly more benefit in increased lifetime than subsequent transitions, as seen in $T_{14}$ where the benefit reaches 72%. This is because later transitions work with deployments where all nodes
are less fresh than their predecessors, and therefore there is less scope for finding such substantial increases.

- Some adaptations generate limited benefit, such as T8 and T9 where the lifetime increase is less than 3.3%. This is because the topology provides few options for extending the lifetime around the nodes that are predicted to fail. In T8 and T9, acquisition nodes fail first, and proactive adaptations reduce the number of operators placed upon these nodes. However, the relocated operators are placed on other acquisition nodes that do not have substantially greater energy reserves. Fundamentally, the original QEP is an effective QEP for the topology, and therefore only small improvements can be gained from proactive adaptations.

The main conclusion from this experiment is that the switch times chosen by the successor of plan transitions were indeed suitable for lifetime extension given the selected QEPs used by each sequence of plan transitions. The transition’s switch times were suitable for maximising the overall lifetime of the deployment, with an average difference from the optimal of 0.43% (computed from Figure 4.6).

4.4 Discussion

Overall, the experiments presented in this chapter show that there is a distinct trade-off between lifetime and continuity of results. If an end user wishes to get the best BFB value for a deployment, it is likely that a proactive approach, integrated into a SNQP as a sequence of plan transitions, will deliver a significant improvement over a non-adaptive SNQP.

If the end user is more focused on the continuity of results, as opposed to the best BFB value for a deployment, then the decision is more complex. Assuming
a very frequent acquisition rate and little tolerance to gaps in the result stream, then it is likely that one of the reactive strategies would be the more suitable choice as it is less likely to cause breaks in the stream than a proactive strategy. However, if there is no tolerance at all to interruptions in the output stream, then neither strategy will perform better than a non-adaptive SNQP unless the acquisition rate is infrequent enough to support the exploitation of the *golden zone* in the agenda. It is worth noting that, if the *golden zone* can be exploited, then it is often better to select the proactive strategy over a completely reactive strategy as both cost approximately the same amount of adaptation time, but the proactive case results in an increased overall functional lifetime.

### 4.5 Related Work

As discussed in Section 1.3, SNQPs, in practice, support little or no adaptive behaviour. SNQPs that compile/optimise a QEP for a given deployment (e.g., [32]) follow a *compile/optimise once, execute many times* paradigm and therefore any differences in the workloads of participating sites has significant effects on the estimated lifetime of the deployment. SNQPs that optimize the QEP for the current state of the deployment (e.g., TinyDB) support load re-distribution by considering the variances in link quality between nodes, but energy reserves are not considered. As discussed in Section 1.3, SNQPs that optimize the QEP at runtime are prone to unpredictable performance, and therefore any lifetime improvement of a QEP cannot be solely attributed to the load redistribution process. Overall, there is no known directly related work in which the QEPs are actively adjusted to account for the different energy demands over the nodes in the deployment.
4.5.1 Routing Based Techniques

By using a SNQP which compiles and optimises a query \( q \) into a QEP for a given deployment structure (e.g., SNEE [32]), it is possible to plan a set of adaptations at compile time that maximises the lifetime of \( q \). This in turn makes runtime load redistribution unnecessary and possibly counter-productive, due to the overhead that runtime decisions incur. For example, consider routing proposals that adjust the site on which an in-network operator is executed, [13, 19, 63, 107], or that reroute tuples along different paths between source nodes and the base station [62, 86, 94] in such way as to avoid overloading a given site. For all these systems, energy is spent at the nodes in carrying out the processing necessary to reach a decision, on top of communication energy. The processing overhead is devoted to updating routing tables or to the selection of better site placements. The overheads have a detrimental effect on the lifetime of the query.

Note also that local decisions are based on limited information and therefore may not be globally beneficial, or else can result in adjustments which keep oscillating between two different states and therefore spend more time adapting than executing tasks. For example, hierarchical clustering techniques [56, 76, 126, 89, 10, 124, 71] execute run-time load redistribution through the periodic selection of new cluster heads. Most of these selection processes do not consider the effect that changing the cluster head of a specific cluster may have on other cluster heads in the deployment. These exiting cluster heads may need to boost their transmission power to relay tuples to the newly-selected cluster head, which would have a detrimental effect on the lifetime of the old cluster heads and possibly on the lifetime of \( q \), overall.

In comparison, a sequence of planned transitions is determined on the basis of global information so that the transitions result in changes that are beneficial
for the entire deployment, whilst actively avoiding cyclic behaviour.

4.5.2 Plan Based Techniques

In other streamed query processing applications, energy expenditure is not a concern. The amount of memory used and the response time are the most important concerns, and therefore adaptive proposals target these aspects of a QEP that can improve either/both metrics, e.g., changes in the algorithm that implements a logical operator [29], or in the MAC protocol [119, 37, 91], or in the execution order of certain operators [83, 117, 8]. In the SNQP context, memory consumption and response time are lesser concerns, as their effect on the energy cost of a given QEP is limited.

4.5.3 Redundancy-Based Techniques

For the majority of the WSN community, a deployment is viewed as a data collection system with little or no in-network processing. This means that concepts such as coverage (how many sensors are actively taking readings in a given geographical area) can be adjusted to support a degree of node redundancy [17, 121], which in turn can be exploited to improve the lifetime of the deployment. The concept of coverage has no meaning in a SNQP context because every node that can contribute to a given extent is expected to contribute at every possible opportunity, and therefore these proposals tend to be incompatible with the SNQP concept.
4.6 Conclusion

This chapter has described a technique for constructing sequence of plan transitions that aims to extend the functional lifetime of a deployment in the face of predictable node failure events, i.e., those that result from energy depletion. The chapter demonstrates that it is feasible to generate a sequence of plan transitions, and that by doing so, the lifetime of a given deployment can be extended, therefore improving the overall BFB value for the deployment.

As previously mentioned, there is no previous work in the SNQP literature that focuses on proactively adjusting QEPs during runtime in order to account for non-uniform energy consumption in participating nodes. There exist techniques for adapting once a node fails due to energy depletion or for other reasons. However, the experimental evaluation has shown that this can limit the potential gains if the node that fails is a critical one, e.g., a sensing node. The closest significant related work is based on clustering algorithms that adjust cluster heads based on residual energy reserves. However, no consideration is given to query processing in such environments. There is also little work on handling clusters when multi-hop communication is prevalent.

A sequence of plan transitions does have limitations, in that it cannot preclude the possibility of (permanent or transient) edge failure. The static version of the SNEE agenda does not adjust for varying rates of tuple delivery during communication events, therefore nodes keep the radio on for the entire communication slot regardless of the number of received tuples.

Note that all the strategies discussed so far consider the functional lifetime to be the functional lifetime of the first acquisition node to fail. This limits the potential for lifetime increases because in these strategies there is no option for replacing an acquisition node. In the next chapter, resilience is reconsidered as a
4.6. CONCLUSION

dimension in deployment design leading to the development of a general technique that can exploit planned redundancy. By exploiting such redundancy, it becomes possible to ensure a minimum degree of resilience in the face of both types of un\textit{predictable failure events} for all types of nodes (acquisition or otherwise).
Chapter 5

Resiliently Designed Deployments

This chapter presents a technique designed to prolong the lifetime of a deployment when both nodes and communication channels can experience both unpredictable node failure events and unpredictable communication channel failure events. In Chapter 3, two reactive techniques were described that operated on deployments where there might be some degree of unplanned redundancy in nodes, and communication channels. However, designing a deployment in which planned redundancy is introduced in order to achieve resilience allows for techniques that capitalize on the designed redundancy to ensure a degree of resilience to both unpredictable node failure events and unpredictable communication channel failure events.

5.1 Motivation

In Chapters 3 and 4, the runtime reprogramming of nodes was seen to be the costliest task in adaptations. Therefore, reducing the need for reprogramming
benefits the end user by reducing the overall time taken by adaptations.

By considering resilience in the deployment design process, it becomes possible to devise techniques that can ensure minimum degrees of resilience $k_n$ and $k_e$ to unpredictable node failure events and unpredictable communication channel failure events respectively.

By using strategies that capitalize on planned redundancy and generate QEPs that intrinsically display given degrees of resilience to both unpredictable node failure events and unpredictable communication channel failure events, the hypothesis is that the performance in terms of the number of tuples delivered to the end user over the lifetime of the deployment can be extended beyond what is possible with adaptive strategies that do not exploit such redundancy.

5.2 K-Resilient QEPs

The ADSNEE infrastructure described in Chapter 3 can be modified to produce K-resilient QEPs. The basic idea is to use the notion of a logical overlay network (defined in Section 5.3) to take advantage of planned (or fortuitous) redundancy in the physical network, and therefore to provide a degree of resilience to failure events.

Figure 5.1 depicts the architecture of the extended version of SNEE, referred to as RSNEE, that is capable of generating k-resilient QEPs when there is sufficient planned redundancy in the deployment for the corresponding logical overlay network to be formed. The process is basically as follows. At compile time, a logical overlay network is computed over the physical deployment (see Stage 1.2 in Figure 5.1). A logical overlay network consists of logical nodes, each of which maps to $k_n$ equipotent nodes, where $k_n$ is the desired resilience level.

The basic idea, therefore, is to derive an equivalence class over the physical
nodes in the routing tree associated with a QEP, and then assign each site-specific task in a QEP to a logical node that maps to a set of equipotent physical nodes (rather than to a single one). In this way, for the QEP to be irreparably compromised due to the failure of a node \( fn \) running a site task \( st \), all the \( k_n \) nodes in the logical node must also have failed. In this way, when \( n \) fails, one of the nodes in the logical node is selected to take over the responsibility for running \( st \) (see Stage 2.3 in Figure 5.1).

These changes give rise to a k-resilient version of SNEE, referred to as RSNEE. The adaptive aspects of RSNEE have manifestations at both compile and runtime. RSNEE can handle both unexpected node failure events and unexpected communication channel failure events.

The communication channels of a WSN are inherently unreliable and unpredictable due to factors such as external interference, the inexpensive radio hardware typically used and the environment the nodes operate in [129, 70], which cause communication packets to be lost. We call unpredictable communication channel failure events the occurrence, broadly speaking, of packet loss. Because these events are often intermittent, it is unlikely that a centralised approach can detect, plan and execute an adaptation in time to effectively mitigate the effects of such events. There is also the risk that such an artefact would end up constantly adapting back and forth and therefore spend more energy and time adapting than actually operating on the task it was assigned. Therefore, to create QEPs that are resilient to unpredictable communication channel failure events in an efficient manner, an approach is needed that adapts quickly and inside the WSN.

In this case, a simple protocol is used that exploits the structure of the logical overlay network by using the \( k^2 \) links between the physical nodes in a logical node and the physical nodes in its logical parent. By using \( k^2 \) links, the protocol attempts to ensure that each packet that is transmitted by a child logical node
5.2. K-RESILIENT QEPS

is received by its parent logical node, while ensuring that the QEP does not fail to meet the delivery time QoS expectation.

The basic idea of the unexpected communication channel failure events aspect of this contribution is to abstract the use of the $k^2$ links to represent communication between logical nodes through logical edges. The protocol, therefore, supports up to $k_e$ attempts for a packet to be received by the parent logical node, where $k_e$ is the number of physical nodes in a logical node that participate in the protocol. Note that $k_e$ cannot be larger than $k_n$, as this would imply that more nodes participate in the protocol than are expected to be available to do so, but can be the same as, or smaller than, $k_n$.

Figure 5.1: The Infrastructure for RSNEE.
5.2.1 Modifying a Deployment for Planned Redundancy

Any deployment can be modified to capitalise on techniques that assume a built-in level of redundancy (such as these based on logical overlay networks). Physical nodes can be added to the deployment layout in such a way that several physical nodes are functionally equivalent to each other. In the example deployment $D$, depicted in Figure 5.2(a), one can make $D$ support a level of redundancy $k_n$ by placing $k_{n-1}$ extra nodes $EN$ geographically close to each node $dn$ in $D$ (except the base station), where each $en \in EN$ and $dn \in D$ share the same functional properties e.g., same sensing capability, memory capabilities and communication capabilities (therefore supporting the same connectivity). This results in a deployment where each physical node has $k_{n-1}$ equivalent counterparts, represented in Figure 5.2(b), where $k_n = 2$, and each redundant node is represented by a dotted circle. Note that Node 6 has been given $k_n$ extra nodes (26 and 27). This is to support the description of aspects of the logical overlay network generation in the next section.

Also note that because the base station, in this form, is a single point of failure. It is assumed throughout the evaluation that it cannot fail. Even though this is an unrealistic assumption, the inclusion of the base station into the technique is trivial, yet it assumes that the system processing the data from the WSN can handle the handover.

5.3 Resilience Through Logical Nodes

This section describes how RSNEE operates and how it generates a logical overlay network over a deployment $D$ which has a built-in level of redundancy $k_n$. The generation process for a logical overlay network occurs at compile time, and can be broken down into four stages, as illustrated in Figure 5.3. These stages can
5.3. RESILIENCE THROUGH LOGICAL NODES

(a) Original deployment’s connectivity graph
(b) Modified deployment’s connectivity graph with a designed redundancy level ($k_n = 2$)

Figure 5.2: Initial and modified deployments.

be summarised as:

- The generation of a coarse-grained overlay network, referred to as a *super-overlay network*, through the use of an equivalence relation and the RT generated by the SNEE compilation stack.

- The generation of a collection of fine-grained overlay networks $[lon_1, \ldots, lon_n]$, each of which is referred to as a *logical overlay network* where each logical node $ln$ contains at least $k_n$ physical nodes.

- The assessment of $[lon_1, \ldots, lon_n]$ to determine the *logical overlay network* that has the longest estimated lifetime.
• The copying of the site tasks from each physical node \( p_n \) that participates in QEP execution to all the physical nodes that share the same logical node \( l_n \) with \( p_n \).

The next four sections describe each phase in detail, using the example deployment connectivity graph in Figure 5.2(b).

### 5.3.1 The Generation of Super-Overlay

The generation of super overlays corresponds to Stage 1 in Figure 5.3, and requires as input:
5.3. RESILIENCE THROUGH LOGICAL NODES

- The degree of desired resilience $k_n$ to be exhibited by the *logical overlay network*.

- A Boolean flag that controls whether acquisition physical nodes should be considered in the super-overlay generation (see below).

- A QEP generated for a given query $Q$ by the **SNEE** compilation stack.

Each node in the RT is referred to as an *active node* (in the sense that, by default, it runs the assigned binary when the QEP starts executing). A super-overlay is generated over the set $CN \subset AN$ of active nodes in the RT, that we refer to as *candidate nodes* for replication. A non-acquisition node in $AN$ is a candidate node. An acquisition node in $AN$ is a candidate node if it is asserted, in the physical schema, to be equipotent to another node in the WSN and if the input Boolean flag is *True*. This caters for the situation in which both the location and sensing electronics of an acquisition node admit a degree of variation, and, therefore, if a specific acquisition node fails, others may be equipotent to it despite being in a different location and possibly having different sensing electronics.

A node $w \in W$, where $W$ is the set of all nodes in the WSN that are not already participating in the QEP, is in the same super-overlay node as an active node $c \in CN$ if the following conditions are met:

- The available memory in $w$ is at least the same as the available memory in $c$ so that $w$ can host the *site task* binary that has been assigned to $c$ by **SNEE**;

- If $c$ is an acquisition node, then $c$ and $w$ have been asserted to be equipotent. This is illustrated in Figure 5.4 with nodes with double circles where the outer circle is dotted.
If $lp(c)$ and $lc(c)$ denote, resp., the parents and children of $c$ in the super-overlay, then there are edges in the connectivity graph between $w$ and, on the one hand, the active node in $lp(c)$, and, on the other hand, the active nodes in $lc(c)$.

In the example deployment connectivity graph (illustrated in Figure 5.2(b)), Nodes 8 and 10 satisfy these conditions and therefore form a super-overlay node, as do Nodes 1 and 14, 2 and 16, 5 and 18, 7 and 24, 3 and 22, 4 and 20, and Node 6 with Nodes 26 and 27. This results in the super-overlay in Fig. 5.4.

5.3.2 Generating Logical Overlays

When a node $w$ belongs to a super-overlay node $\Omega$, the conditions above ensure that $w$ can communicate directly with the active node (as opposed to every node) in its children and its parent super-overlay nodes in $\Omega$. The conditions above do
not ensure that $w$ can communicate directly with every other node (as opposed to the active one) in its children and its parent super-overlay nodes in $\Omega$. There are several permutations in which the physical nodes belonging to two logical nodes can communicate with each other. In order to determine the best permutation of physical nodes in each logical node, the complete set of valid permutations is generated, giving rise to a set of candidate logical overlays for assessment.

The algorithm that carries out this assessment starts by traversing the RT from the root (the base station) until it reaches a node for which a super-overlay node has been formed. All the connecting permutations between the physical nodes in each of the logical nodes are considered, with a view to forming logical nodes with a minimum size $k_n$, which denotes the required resilience level. Once the parent permutations are generated, the process is repeated, and the children permutations are generated, and so on.

In both cases, the correctness condition is that there are $k_n$ physical nodes
in every logical node and that, respecting the RT topology, two logical nodes form a logical edge, i.e., that each physical node in one logical node can communicate directly with each physical node in the other logical node. For every such permutation, a candidate logical overlay node is generated, and the process proceeds down the RT. If no correct permutation can be generated, the parent permutation is removed from consideration and a new parent permutation is considered. If there are no correct permutations left to assess, the algorithm returns a failure-to-form-overlay result, as a logical node with resilience level $k_n$ could not be formed for that RT.

In the example, assuming that $k_n = 2$, and that equipotent acquisition nodes have been asserted, Node 8 would be located first. The permutation consisting of Node 10 and Node 8 would be generated as it is the only correct permutation, if $k_n = 2$. The same behaviour occurs with Node 8’s children (i.e., Nodes 1, 2 and 5), as their logical nodes can only generate one correct permutation, for $k_n = 2$. The same can be said for Node 2’s children (i.e., Nodes 4 and 3). Node 5’s child (i.e., Node 6) results in three different permutations, which are: [6,27], [6,26], and [6,26,27], each of which satisfies $k_n = 2$, and therefore three different candidate logical overlays are now considered.

When faced with Node 7, there is again only one correct permutation if $k_n = 2$. However, because Node 26 is unable to communicate directly with Node 24, the two candidates stemming from Node 6 which contain node 26, ([6,26] and [6,26,27]) are rejected, resulting in the generation of the single complete logical overlay network with $k_n = 2$, illustrated in Figure 5.5.
5.3. RESILIENCE THROUGH LOGICAL NODES

5.3.3 Assessing The Logical Overlay Set

The set of $k_n$-resilient logical overlays is assessed to determine which element in it leads to the longest estimated functional lifetime for the QEP, assuming that nodes only experience the predictable node failure event of energy depletion. The SNEE energy cost model [16] is used to determine the sequence in which nodes fail, interleaved with adaptations, until the logical overlay network cannot support future adaptations. In this way, we can estimate the lifetime of the QEP over a sequence of predictable node failure events. The logical overlay that leads to the best estimated lifetime for the QEP over a sequence of predictable node failure events is then selected.

5.3.4 Replication of Site Tasks

The last compile-time step is to assign the site task of the active node in each logical node $\Omega$ to all the $k_{n-1}$ equipotent physical nodes in $\Omega$. These nodes start
in the inactive state, i.e., only the node in the original RT is active at the start of QEP execution. In the running example, this results in the assignments depicted in Figure 5.6.

Assigning *site tasks* at compile time rather than runtime removes the requirement for runtime adaptations to reprogram any nodes, thereby removing the most time- and energy-consuming task in runtime adaptations.

### 5.3.5 Adapting to *Unpredictable Node Failure Events*

At run time, when an active node $a$ experiences an *unpredictable node failure event* $e$, if $a$ has a set $I$ of equipotent inactive, non-failed nodes, then the adaptation strategy selects the node $i \in I$ with the smallest average cost for communicating with $lp(a)$ and $lc(a)$. Then, $i$ is removed from $I$ and becomes the active node, $a$ is added to the set of failed nodes, and the children of $a$ are reassigned to become
the children of \( i \) (i.e., their parent edges are redirected from \( a \) to \( i \)).

At run time, this adaptive response leads to the following types of messages being sent and responded to: (a) redirection messages, which change the destination node for a node’s messages, and (b) activation messages, which cause a node to start executing their agenda. In the example, this means that, at runtime, if Node 8 fails, since \( I = \{10\} \), Node 10 receives an activation message and becomes the active node (and \( I = \{\} \)). Also, Nodes 2, 1 and 5 receive a redirection message and start sending tuples to Node 10. As a result, the actively executing DAF for the query becomes the one depicted in Figure 5.7.

### 5.4 Resilience Through Logical Channels

Note that, so far, no logical abstraction is in place over physical edges. This section describes how RSNEE can be augmented to adapt to unpredictable communication channel failure events. The basic idea is to define logical channels between logical nodes, by abstracting the communication between logical nodes in a logical overlay network to be through logical channels instead of physical channels. This results in a minimum of \( k_e \) physical channels having to experience unpredictable communication channel failure events for a packet to be lost. Note that \( k_e \) must always be lower than or equal to \( k_n \). This is because only nodes that participate in a logical node can participate in the associated logical channels.

The process of generating a general logical overlay network that is robust to unpredictable communication channel failure events is illustrated in Figure 5.8. The first four stages of the compilation stack are identical to that of the generation of a logical overlay network for unexpected node failure events except that the equivalence relation used in the generation of the global overlay network is modified to be stricter through an additional requirement as follows. Recall
Figure 5.8: The compilation and runtime stack from the RSNEE when robust to unpredictable communication channel failure events.

the definition and notation in Section 5.3.1 above, then

- If $LN(c)$ denotes the current physical nodes in the logical node for $c$ in the super-overlay, and $lpn \in LN$, then $lpn$ and all nodes in LN can communicate directly.

The reason for requiring all nodes within a logical node to communicate with each other directly is that, now, physical nodes in a logical node are expected to overhear each other’s transmissions. It is expected that when two or more nodes $n$ and $n'$, with the same communication technology, can communicate with two or more other nodes $m$ and $m'$, that $n$ and $n'$ can communicate with each other, as illustrated by Figure 5.9. Note that an obstacle could be placed directly between $n$ and $n'$ or they may have different transmission technology, but these cases are not considered in this thesis.

Stage five of the extended compilation stack in Figure 5.8 is the integration of a simple communication protocol that exploits the $k!$ physical channels between
5.4. RESILIENCE THROUGH LOGICAL CHANNELS

Figure 5.9: A graphical representation of the ability for nodes within a logical node to communicate with each other.

the physical nodes $pn \in \Omega$ and the active physical node $apn$ of $\Omega'$, where $\Omega'$ is the parent logical node of $\Omega$. The protocol is designed to ensure that tuples arrive at the base station within the QoS for delivery time set for the query. The next two sections describe:

- the overall motivation behind the communication protocol;
- how the protocol abstracts the physical communication tasks in a QEP;
- how the protocol operates when faced with an unexpected communication channel failure event at runtime.

5.4.1 Motivation

The SNEE SNQP is mainly focused on generating QEPs that always meet the specified QoS expectations for acquisition interval and delivery time (as discussed in Section 2.3). Some routing protocols (e.g., TCP/IP [108], RIME’s uip [27]) focus on ensuring completely reliable communication channels on a hop-to-hop
basis, with little or no consideration to the period of time required to ensure such reliability. This often makes such routing protocols incompatible, on their own, with QEPs that need to ensure they meet the overall delivery time QoS specified by the end user. These protocols are often used in conjunction with a load-shedding technique [80] that determines when and which tuples to drop, thereby avoiding running out of memory if backlogs build up.

In contrast, the protocol proposed here is designed to ensure that tuples are delivered to the end user within the specified delivery time. However, it cannot ensure that all tuples are delivered because of the limited time frame. Delivery time constraints are accommodated using a globally planned set of timed transmissions that are used to overcome packet losses incurred when physical channels experience unpredictable channel failure events.

5.4.2 The Communication Protocol

The robustness of the protocol comes from exploiting the broadcast nature of wireless transmission by having some nodes overhear transmissions and retransmit them if needed. The active node of the receiver can distinguish between copies of packets it already has received and packets lost from a previous transmission, through the use of a packetID that is transmitted with all but the first transmission packet. The first transmission packet contains a packetCount, instead of a packetID, that informs the receiver how many packets it should expect to receive.

5.4.2.1 Modifying a QEP Agenda

Stage 5 in Figure 5.9 replaces the QEP agenda with a logical agenda that contains sets of physical transmission tasks that together represent one logical transmission
task. Instead of using just the active node in each logical node to transmit/receive packets, a logical channel uses up to $k_c$ physical nodes in the source, the set of which is referred to as the sibling nodes, to transmit the packets to the active node in the target. These additional tasks are added with the assumption that not all of them may be needed. These additional tasks are added for each communication event between two logical nodes, or between a logical node and the base station, and are scheduled so that the agenda still completes within the specified delivery time. Figure 5.10 presents the pseudocode for generating the logical agenda.

**BUILD-LOGICAL AGENDA**($R_Q$, $\alpha$, $\beta$, Duration, $LON_Q$)

▷ schedule leaf fragments first
1 for $i \leftarrow 1$ to $\beta$
2 do for each $s \in R_Q.Sites$
3 do $nextSlot[s] \leftarrow \alpha \ast (i - 1)$
4 while
▷ post-order traversing active nodes in $LON_Q$
▷ let $f$ denote the current fragment
5 do if $f.IsLeaf = yes$
6 then $s.f.ActAt \leftarrow []$
7 $s \leftarrow f.SiteID$
8 do $s.f.ActAt.Append(nextSlot[s])$
9 $nextSlot[s] \leftarrow + \text{Duration}(s,f)$
▷ schedule non-leaf fragments next
10 while
▷ post-order traversing $R_Q$,
▷ let $s$ denote the current site
11 do while
▷ post-order traversing active nodes in $LON_Q$
▷ let $f$ denote the current fragment
12 do if $f \in s.AssignedFragments$
13 then $f.ActAt \leftarrow nextSlot[s]$
14 $nextSlot[s] \leftarrow + \text{Duration}(f)$
▷ schedule comms between logical nodes
15 commBetweenLogicalNodes($LON_Q$, $s$, $s.parent$, Duration)
16 return agenda

Figure 5.10: An algorithm for computing a logical agenda.
The pseudocode in Figure 5.10 is almost identical to the agenda construction code in Figure 2.14, in that given a routing tree, an acquisition rate, a buffering factor, the length of time the query is set to execute for and a logical overlay network, returns a timed agenda for the query that takes into account the nodes within a logical node. The difference is in Line 15, which in the present case calls a function to add the timed tasks corresponding to a logical transmission. Figure 5.11 presents pseudocode for inserting the physical transmissions that together represent a logical channel transmission. An initial transmission (Lines 2-5) takes place between the active nodes \( cpn \) and \( ppn \) in \( \Omega \) and \( \Omega' \), where \( \Omega' \) is the parent logical node of \( \Omega \).

The first acknowledgement stage (Lines 6 - 10) takes place: \( ppn \) can transmit an acknowledgement packet to the active physical nodes of \( \Omega \) (which, at this point, is timed to be both \( cpn \) and its sibling nodes) and the active physical node of \( \Omega \) and its sibling nodes are timed to receive the acknowledgement packet. The power of the acknowledgement transmission is set to that needed to reach the furthest active node in \( cln \) and its sibling nodes. This is because the nodes that require lower energy levels to receive packets from \( ppn \) are able to overhear this transmission, due to the broadcast nature of wireless communication, and thereby \( ppn \) saves energy by only needing to transmit the packet once instead of through \( k_e \) different channels.

Line 12 obtains the sibling nodes of \( \Omega \) in order of transmission from/to each sibling node and \( ppn \). The order is important because, as illustrated in Lines 13 - 27 in Figure 5.11, each sibling is given its own timed slot to transmit the packets to \( ppn \) and to the rest of the sibling nodes that are active in its given timeslot. By ordering the transmissions, the cheaper ones can be used first therefore saving energy if the more expensive nodes are not needed.
COMM BETWEEN LOGICAL NODES($LON_Q$, activeChild, activeParent, Duration)
1. activeSiblings ← $LON_Q$.getActiveSiblings(activeChild)
   ▷ schedule initial comm between active nodes in logical nodes
2. activeChild.TX(activeParent).ActAt ← max(nextSlot[activeChild], nextSlot[activeParent])
3. activeParent.RX(activeChild).ActAt ← activeChild.TX.ActAt
4. nextSlot[activeChild] ← + Duration(activeChild.TX)
5. nextSlot[activeParent] ← + activeParent.RX
   ▷ schedule initial acknowledgement stage
6. activeParent.TXA(furthestAway(activeSiblings) ∪ activeChild).ActAt ← max(nextSlot[activeChild], nextSlot[activeParent])
7. for each $s$ ∈ (activeSiblings ∪ activeChild) do
8.     activeChild.RXA(activeParent).ActAt ← activeParent.TXA.ActAt
9.     nextSlot[$s$] ← + Duration($s$.RXA)
10.    nextSlot[activeParent] ← + activeParent.TXA
11. ▷ schedule redundant transmissions
12. activeSiblings ← [(activeChild, $LON_Q$.siblingsInRankedOrder())]
13. while activeSiblings.size() ≠ 0 do
14.     currentActiveSibling ← activeSiblings[0]
15.     activeSiblings.remove(currentActiveSibling)
16.     currentActiveSibling.TX(furthestAway(activeSiblings ∪ activeParent)).ActAt ← max(nextSlot[currentActiveSibling ∪ activeSiblings], nextSlot[activeParent])
17.     for each $s$ ∈ (activeSiblings ∪ activeParent) do
18.         s.RX(currentActiveSibling).ActAt ← currentActiveSibling.TX.ActAt
19.         nextSlot[$s$] ← + Duration($s$.RX)
20.     nextSlot[currentActiveSibling] ← + Duration(currentActiveSibling.TX)
21. ▷ schedule any redundant acknowledgements
22.     if activeSiblings.size() ≠ 0 then
23.         activeParent.TXA(furthestAway(activeSiblings)).ActAt ← max(nextSlot[activeSiblings], nextSlot[activeParent])
24.     for each $s$ ∈ activeSiblings do
25.         s.RXA(activeParent).ActAt ← activeParent.TXA.ActAt
26.         nextSlot[$s$] ← + Duration($s$.RXA)
27.     nextSlot[activeParent] ← + activeParent.TXA

Figure 5.11: An Algorithm for adding redundant tasks that together represent a logical channel.
The packets transmitted to each sibling are the ones received during the previous sibling and active node transmissions for this logical transmission. After each sibling has completed its transmission, it goes into a low-powered sleep mode, because its participation in the logical transmission has now finished (this is represented by its removal from the sibling collection in Line 15 in Figure 5.11).

Another timed acknowledgement transmission/reception task is added between each sibling transmission (Lines 21 - 27 in Figure 5.11). Note that this only occurs when there are siblings that are expected to transmit received packets. This saves energy because once a sibling receives an acknowledgement packet, it puts itself into a low powered state and ignores future timed transmission tasks for this logical channel transmission.

The overall result is a logical agenda, where logical channels arise from several timed, potentially redundant physical transmissions. The logical agenda for the QEP presented in Figure 5.6 is illustrated in Figure 5.12 where the tasks highlighted in red (between time slots 300100 and 3301304) represent a logical transmission between logical nodes 7 and 6.

By generating a logical agenda at compile time, the logical agenda can ensure that under most circumstances packets are delivered to the end user within the delivery time QoS expectation. The decision as to when a task is redundant is controlled by the runtime protocol described in the next section. Note, that it is possible for each logical transmission to include many physical transmissions/receptions and still meet the delivery time QoS expectation if the golden zone, defined in Section 3.4.2.4, supports multiple iterations of the tasks inserted in Lines 11 - 27 in Figure 5.11. Currently, this is not supported and is left for future work as discussed in Section 6.3.4.
### 5.4. RESILIENCE THROUGH LOGICAL CHANNELS

| Time (ms) | 0 | 100 | 300000 | ... | 3300000 | F2,1 | F2,2 | F1,1 | F1,2 | F2,1 | F2,2 | ... | F2,11 | F2,11 | ... | F1,11 | F1,11 | ... | F2,11 | F2,11 | ... | ... | ... | ... |
|-----------|---|-----|--------|-----|--------|------|------|------|------|------|------|-----|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sites     | F2,1 | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 |
| 0         | 7  | 24  | 6     | 27  | 5     | 18   | 8     | 10   | 1     | 14   | 2     | 16   | 12  | 16 | 20 | 20 | 18 | 14 | 10 | 6   | 2   | 8   | 5   | 12   | 16   | 12   | 16   | 20   | 18   | 14   | 10   | 6   |
| 100       | F2,1 | F1,1 | F2,2  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 | F2,1  | F2,2 | F1,1  | F1,2 |
| 300000    | ... | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... | ...   | ... |

**Figure 5.12:** Representation of the agenda after the redundant transmission tasks are added.
5.4.2.2 Runtime Tasks

During the execution of the logical agenda, a runtime protocol is run on each logical node that determines if a transmission/reception task scheduled in the logical agenda is to take place or not. Figures 5.13 - 5.18 use the model and notation for distributed algorithms described in [4]. Figure 5.13 describes the initial set up of variables for the protocol. There are three possible states to a node: OPERATING, LISTENING and SLEEPING. The protocol is valid under the following assumptions:

- Communication channels are bidirectional (i.e., data can travel in both directions). This is a valid assumption for WSNs because it is assumed that motes have the same transmitter/receivers capabilities.

- The clocks on all the motes that participate in the protocol are synchronised. This assumption does not hold automatically in real deployments, where motes can become unsynchronised over time, but such circumstances are not covered in this thesis, and there is evidence [32] that clock drift is not as significant a factor as one might intuitively suppose. It is also worth noting that there are techniques in the literature [102, 55, 100, 105] which could be used to compensate for clock drift.

- The radio broadcasts messages, i.e., all channels that can be reached through the same or lower transmission power is indeed reached.

- All sites have executed all the acquisition fragments for a given agenda cycle.

Each node knows (i.e., is programmed with) its parent $p\Omega \text{ id}$, its children ids ($c\Omega$) and its own $\Omega \text{ id}$.
$P_{INIT} \equiv "X \text{ entities have tuples to transmit to the base station at time } t" \equiv \\
\exists x, y \in N \text{ that } value_t(x) = 1 \land y \neq x \land value_t(y) = 0 \land isBase(y),$ \\
$P_{FINAL} = "The base station y has all tuples at time } t" \equiv \\
\exists x \in N \text{ that } value_t(x) = 1 \land isBase(x)$

$S = \{ \text{OPERATING, LISTENING, SLEEPING } \}; \\
S_{INIT} = \{ \text{SLEEPING} \}; \\
S_{TERM} = \{ \text{SLEEPING} \}. \\
S_{REGISTERS} = \{ \text{logicalNodeID, outputLogicalID, inputLogicalIDs,} \\
\text{ inputPackets[][], maxPackets[], acks[], outputPackets[] } \\
\text{ allPackets, physicalID, isactiveNode } \}$ \\
$R = \{ \text{Bidirectional Links, Unitary communication delays,} \\
\text{ Synchronized clocks, Broadcast radio, Acquisition executed} \}$

Figure 5.13: The initial set-up for the runtime protocol.

All sites start in the SLEEPING state, representative of when all sites have finished executing all acquisition fragments. Figure 5.14 describes the runtime protocol logic for when sites are in the SLEEPING state. No behaviour is scripted to happen spontaneously when a site is sleeping, because it is in a low power state, saving energy. As mentioned previously, everything is controlled by the agenda, and therefore timed agenda tasks can cause a mote to move out of the SLEEPING state to either the OPERATING or LISTENING state. The decision about which state to move to is based on the type of agenda task timed to execute, whether the site has received an acknowledgement packet from $p\Omega$ within this agenda cycle, or whether the site has transmitted an acknowledgement packet to the $c\Omega$ to which the task is associated with. Before an agenda cycle repeats (recall that the agenda repeats periodically until either the QEP fails or a stop message is sent into the network), the received acknowledgement flags $acks[]$ are set to false and the packet stores are emptied.

Note that if the received acknowledgement flag in a site is set to true, when a timed agenda task is scheduled to execute, then the site ignores the task and
SLEEPING
1   Spontaneously → nil
2   Received(P) → nil
3   when agenda task (A) scheduled to start
4   begin
5     If A instance of RX(target) task \ target = anyof(inputLogicalIDs) ∧ !acks[logicalID(target)] Then
6     become LISTENING(A).
7     If A instance of RX(target) task \ target = !anyof(inputLogicalIDs) ∧ !acks[outputLogicalID] Then
8     become LISTENING(A).
9     If A instance of RXA(target) task \ !acks[outputLogicalID] Then
10    become LISTENING(A).
11    If A instance of Agenda Cycle tick Then
12       reset maxPackets, acks, allPackets
13       become SLEEPING.
14     If A instance of fragment task \ ( A instance of TX(target) task \ !acks[outputLogicalID]) Then
15     become OPERATING(A).
16   end

Figure 5.14: The runtime protocol for SLEEPING state.

stays in the SLEEPING state. These tasks are considered redundant once the
acknowledgement flag is set to true, as the site is no longer needed for that logical
transmission.

Figure 5.15 describes the possible actions of the runtime protocol when sites
are in the OPERATING state. If the agenda task that moved the site into
the OPERATING state is a fragment task, then the site executes the scheduled
fragment, places the output packets into the output packet tray and then returns
to the SLEEPING state (lines 5 - 8 in Figure 5.15). Lines 9 - 14 describe the
actions involved if a transmission task is timed to execute on Ω. The site takes
whatever packets are in the output packet tray and transmits them to the active
site of pΩ. All packets are transmitted with a packetID, apart from the first
5.4. RESILIENCE THROUGH LOGICAL CHANNELS

OPERATING

1  Spontaneously $\rightarrow$ nil
2  Received($P$) $\rightarrow$ nil.
3  when agenda task ($A$) scheduled to start
4  begin
5  If A instance of fragment task $\land \neg$doneInNetworkOperations Then
6    DoInNetworkOperators(packets)
7    doneInNetworkOperations = TRUE
8  become SLEEPING;
9  If A instance of TX(target) task $\land$ isactiveNode Then
10     For (packet i to n in outputPackets)
11        If outputPackets[i] != null $\land$ i == 0 Then
12           Send(outputPackets[i], logicalNodeID, |outputPackets|, physicalID) to N(outputLogicalID)
13        ElseIf outputPackets[i] != null Then
14           Send(outputPackets[i], i, logicalNodeID, physicalID) to N(outputLogicalID)
15     become SLEEPING;
16  If A instance of TX(target) task $\land$ !isactiveNode Then
17     For (packet i to n in outputPackets)
18        If outputPackets[i] != null $\land$ i == 0 Then
19           Send(outputPackets[i], logicalNodeID, outputPackets[i].MaxPackets, physicalID) to N(outputLogicalID)
20        ElseIf outputPackets[i] != null Then
21           Send(outputPackets[i], i, logicalNodeID, physicalID) to N(outputLogicalID)
22     become SLEEPING;
23 end

Figure 5.15: The runtime protocol for OPERATING state.

packet, which is transmitted with the number of packets in the output packet tray (when transmitted by the active site of $\Omega$). Lines 16 - 21 describe actions to take when a transmission task is planned to execute on a sibling node of $\Omega$. The sibling node transmits packets received from previous transmission tasks without changing any values. Once a node has transmitted its output packets, the node returns to the SLEEPING state.
Figure 5.16 and Figure 5.17 describe the actions of the protocol when sites are in the *LISTENING* state. Figure 5.16 describes what state changes are to occur, given different, scheduled, agenda tasks. If the task is a fragment task, or a transmission task, then the site moves into the *OPERATING* state, so that it can execute the fragment, or transmit packets to \( p\Omega \). If the scheduled task is a reception task (either *RX* or *RXA*), then the protocol does nothing, as the site is already in the correct state. If the scheduled task is a scheduled acknowledgement transmission task (TXA), then the site executes the *SendAckPacketsToInputs* logic presented in Figure 5.18, which checks to see if it has received all the expected packets from \( c\Omega \), and either sends an acknowledgement packet to the active sites in \( c\Omega \), or does nothing. If the site does transmit an acknowledgement packet, then the site goes to the *SLEEPING* state, as it no longer needs to receive any more input packets from \( c\Omega \).

Note the use of a *radio off* task in Figure 5.16, which does not appear in the agenda in Figures 5.12 and 2.15. The *radio off* task is activated at the end of a communication task, to turn off the radio and put the mote into sleep mode. This saves energy by only having the radio on for the minimum amount of time required to do the transmission. In the logical agenda, this *radio off* task is only activated when there is no communication task scheduled to execute straight after the last communication task.

Only when a site is in the *LISTENING* state can the site receive packets. In the agenda, the only packets that are expected to be received come from its children logical nodes or its parent logical node, but, for completeness, the behaviours from all possible packets received are expressed in Figure 5.17. Lines 3 - 5 in Figure 5.17 describe the reception of an acknowledgement packet from the active site in \( p\Omega \), which results in the site setting an acknowledgement flag to true and then going to the *SLEEPING* state. Lines 6 - 7 in Figure 5.17 describe
5.4. RESILIENCE THROUGH LOGICAL CHANNELS

LISTENING
1. Spontaneously → nil.
2. When agenda task (A) scheduled to start
   begin
   4. If A instance of TX(target) task || A instance of fragment task Then
      become OPERATING.
   6. If A instance of RX(target) task Then
      ▷ do nothing as already in correct state.
   8. If A instance of RXA(target) task Then
      ▷ do nothing as already in correct state.
   10. If A instance of TXA(target) task Then
      SendAckPacketsToInputs(target)
   12. If A instance of RADIO-OFF task Then
      become SLEEPING.
   end

Figure 5.16: The runtime protocol for LISTENING state.

the action of overhearing packet transmission from a sibling, which results in the
site storing the received packet for possible retransmission in the future. Lines
8 - 13 in Figure 5.17 describe the reception of packets from a child node cΩ.
Depending on the type of packet received, it either stores both the packet data
and the number of expected packets to be received, or just the packet data. Lines
14 - 19 represent states that should not occur during normal operations because
of the agenda tasks, but if they were to occur, the node would be put in the
SLEEPING state to save energy.

The rest of this section consists of a walk through of the logical transmission
between logical nodes 7 and 6 in the agenda in Figure 5.12.

First, assume that the agenda in Figure 5.12 and the QEP illustrated in Figure
5.6 have been executing, and have completed the tasks scheduled before and up
to time stamp 3300000. All sites are currently in the SLEEPING state, Site 7
is scheduled to execute the transmission task (TX6) at time stamp 3300100 and
LISTENING

1. \texttt{Received(P)}

2. \texttt{begin}

3. \texttt{If P.sender = outputLogicalID \&\& P instance of ack Then}

4. \texttt{acks[P.sender] = true;}

5. \texttt{become SLEEPING}

6. \texttt{If P.sender = logicalNodeID \&\& P instance of data Then}

7. \texttt{outputPackets[p.packetID] = p}

8. \texttt{If P.sender = anyOf(inputLogicalIDs) \&\& P instance of data Then}

9. \texttt{If P instance of firstDataPacket Then}

10. \texttt{maxPackets[logicalID(P.sender)] = P.packetCount}

11. \texttt{inputPackets[logicalID(P.sender)][0] = P.Payload}

12. \texttt{Else}

13. \texttt{inputPackets[logicalID(P.sender)][P.packetID] = P.Payload}

14. \texttt{If P.sender = outputLogicalID \&\& P instance of data Then}

15. \texttt{become SLEEPING \triangleright Invalid state}

16. \texttt{If P.sender = logicalNodeID \&\& P instance of ack Then}

17. \texttt{become SLEEPING \triangleright Invalid state}

18. \texttt{If P.sender = any(inputLogicalIDs) \&\& P instance of ack Then}

19. \texttt{become SLEEPING \triangleright Invalid state}

20. \texttt{end}

Figure 5.17: The runtime protocol for \textit{LISTENING} state when a packet is received.

Site 6 is scheduled to execute the reception task \textit{RX7}. Site 7 would move from the \textit{SLEEPING} state to the \textit{OPERATING} state, and Site 6 would move from the \textit{SLEEPING} state to the \textit{LISTENING} state. Once in the \textit{OPERATING} state, Site 7 would transmit its output packets.

Assuming that only one tuple can be stored in a packet, then 11 packets would be transmitted, because 11 tuples were buffered (assuming a selectivity of 1 for the acquire operator) between time stamp 0 and 3300000 that need transmitting. The first packet to be transmitted to Site 6 would contain the first tuple, and the packet count of 11. If received, Site 6 would store both the first packet in the input packet tray (at index 0), as well as store 11 in the expected number
5.4. RESILIENCE THROUGH LOGICAL CHANNELS

Figure 5.18: The runtime protocol method SendAckPacketsToInputs.

of packets variable. Each subsequent packet, if received, would be transmitted and stored in the input packet tray using the packet id as the index. After the transmission, Site 7 would return to the SLEEPING state.

An acknowledgement transmission is then scheduled for Site 6 (TX6) and an acknowledgement reception for Sites 7 and 24 (RX7, RX24). This results in Sites 24 and 6 moving from the SLEEPING state to the LISTENING state. Site 6 would then check the expected number of packets against the actual number received, and, assuming that all the packets were received, would transmit an acknowledgement packet to Sites 7 and 24 by broadcasting the packet with the strength needed to be received by the further away of the two sites. After this, Site 6 would move to the SLEEPING state and ignore the scheduled agenda tasks at timestamps 3300702, 3301003 and 3301304.

If the acknowledgement packet was received by Sites 7 and 24, they would go into the SLEEPING state and also ignore the scheduled agenda tasks at time stamps 3300702, 3301003 and 3301304. This transmission behaviour is illustrated in Figure 5.19(a), where horizontal lines represent time, black arrows represent communication between sites, and dashed arrows represent communication task duration.
Figure 5.19: Different transmission behaviours.

If not all the packets were received by Site 6, then it would not transmit an acknowledgement packet, and therefore would stay in the *LISTENING* state. This means that Sites 7 and 24 would not hear an acknowledgement packet, and therefore would stay in the *LISTENING* state.

The next scheduled task pair in the agenda is a transmission and reception pair (*TX7* and *RX6*), and would result in Site 7 moving to the *OPERATING* state and retransmitting its output packets to both Sites 24 and 6. If received, each site would store the packets in either their input tray (Site 6), or its output tray (Site 24). After the transmission, Site 7 goes into the *SLEEPING* state and is not called upon again until the next agenda cycle.

A second acknowledgement transmission/reception task (*TXA24* and *RXA6*) for Site 6 is scheduled to execute. This results in no change in states for either site. Site 6 checks the expected number of packets against the actual number received and, assuming all the packets were received, then it would transmit an acknowledgement packet to Site 24 and move to the *SLEEPING* state and ignore the scheduled agenda tasks at timestamps 3301304.

If the acknowledgement packet was received by Site 24, it would go to the *SLEEPING* state and also ignore the scheduled agenda tasks at timestamps 3301304. This transmission behaviour is illustrated in Figure 5.19(b).
If not all the packets were received by Site 6, then it would not transmit an acknowledgement packet and therefore would stay in the \textit{LISTENING} state. Therefore, Site 24 would also stay in the \textit{LISTENING} state, because it would not have heard an acknowledgement packet.

The next scheduled agenda task pair, is a transmission and reception pair (\textit{TX6} and \textit{RX24}) that results in Site 24 moving to the \textit{OPERATING} state. It would transmit the output packets received from Site 7, to Site 6. After the transmission of the stored packets, Site 24 would go to the \textit{SLEEPING} state as its transmission tasks are complete. At this point, the radio off task is executed for Site 7, which places it in the \textit{SLEEPING} state. This results in the transmission behaviour illustrated in Figure 5.19(c).

\section{Evaluation}

The experiments reported in this section investigate whether the strategy described in Sections 5.3 and 5.4 can increase the functional lifetime of the deployment in the face of a sequence of unpredictable node failure events (presented in Section 5.5.2) and increase the number of tuples delivered to the end user per agenda cycle given unpredictable communication channel failure events (presented in Section 5.5.3).

For both sets of experiments, the following assumptions were made:

1. All metadata has been collected. Metadata collection costs are modest compared with the evaluation of queries over many agenda cycles and therefore are ignored during these evaluations.

2. The probability of a tuple passing any selectivity constraint in the operators is 100\%. This pessimistic assumption places the most substantial energy
consumption burden on the QEP.

3. Query compilation is executed on a host machine that has ample processing and memory capabilities.

4. There is one base station with a connection to mains power, the base station for MicaZ motes can be directly connected to a laptop by a USB cable from which the mote can draw power itself; this is not an unreasonable assumption.

5. When either a running acquisition node fails in a QEP, to which the strategy cannot adapt, or a node fails that results in a partitioning of the RT in such a way that the strategy cannot ensure tuples from all acquisition nodes reach the base station, then the strategy and QEP is deemed to have completely failed.

5.5.1 Topology Generation

Initially, 15 topologies are used as the basis for generating topologies with different levels of designed redundancy. To generate these, five topologies were randomly selected from each query set defined for the small topology scale set described in Sections 3.4.1 and 4.3. Routing trees were generated over these topologies using given queries. Nodes that did not participate in the RT were removed from the topology. To support topologies with different levels of designed redundancy, each topology was changed to that between 0 and 5 extra nodes were inserted into each topology for each RT node $rn$, except for the base station. Each extra node $en$ was assumed to have the same links as its $rn$ counterpart, but with varying energy costs and variation in the ability of the $rn$ to communicate with the $k_{n-1}$ extra nodes inserted for the $rn$ as well as its children and the parent of $rn$. This
5.5. EVALUATION

(a) The reduced deployment topology.
(b) The final topology with a designed redundancy ($k_n$) of two.

Figure 5.20: How to adjust the topologies from previous evaluations.

is representative of the placement of an extra node relatively close to $rn$ (without obstructions) in a real deployment.

For example, assume the topology shown in Figure 2.4(a) is used as the basis for designing a resilient deployment. Now, assume that the RT generated by compiling a QEP using the query and schemas in Figure 2.4(b), is the one in Figure 2.7. This would result in the removal of Node 9 from the original topology, in Figure 5.20(a). Assuming a level of designed redundancy of 2, one extra node would be inserted for every node in the RT. The resulting topology would consist of 17 nodes, e.g., as shown in Figure 5.20(b).

For the experimental results presented in this section, topologies were generated with varying levels of designed redundancy, between 1 (representative of the behaviour of the original SNEE infrastructure where only 1 physical node participates in each logical node) and 5 (selected as a level of designed redundancy that may still remain affordable by an end user in realistic scenarios). Each query set then comes with 25 different topologies, for the variable redundancy levels for a
given deployment layout.

For the communication channel evaluation in Section 5.5.3, each topology was adjusted again, so that five different versions of the same topology were generated where the distances between nodes that participate in different logical nodes were adjusted to lie 8, 16, 24, 32 and 40 meters apart thereby giving rise to different network densities.

Each query was compiled and executed with the first set of QoS expectations defined in Table 3.1, as this shows the behaviour clearest. It is worth noting that this is a general feature, and is not limited to this specific experiment set.

5.5.2 Experimental Results: Unpredictable Node Failure Events

This section investigates whether the use of RSNEE for adapting to unexpected node failure events can increase the lifetime of a query $q$ on a deployment $d$, thereby improving its BFB value. The experiments are divided into three investigations:

- The first aims to determine to what extent the approach described in this chapter and one purely reactive technique, complete re-optimisation as described in chapter 3, lead to increased query lifetimes in relation to the non-adaptive original SNEE infrastructure.

- The second aims to determine the amount of energy used by the adaptation process.

- The third aims to determine the amount of downtime imposed by the adaptations.
The reasoning behind investigating the effect that several *unexpected node failure events* have on the execution of a QEP is that, in real world deployments it is likely that more than one node will experience a *unexpected node failure event*, during QEP execution. As in previous chapters, *unexpected node failure events* are distributed uniformly over the estimated QEP lifetime.

Note that, in this section, communication channels are assumed to be stable, and therefore the communication protocol that responds to *unpredictable communication channel events* has been deactivated for the experiments reported in the section.

### 5.5.2.1 Lifetime Measurements in Unstable Deployments

To estimate the lifetime of a query \( qu \) on a deployment \( D \) subject to *unpredictable node failure events*, a QEP \( q_i \) is generated by inputting \( qu \), physical and logical schemas, into the RSNEE compilation stack. Once \( q_i \) is generated, it is shipped to \( D \). The SNEE energy cost model [16] is used to estimate the lifetime of the QEP by assessing each node in \( D \). The shortest lifetime of a node in the QEP is the time in which *unpredictable node failure events* is estimated to occur.

The decision to use the number of agenda cycles that have occurred before the first *predictable node failure event* occurred as the latest point at which *unpredictable node failure events* can occur was made because using a larger metric opens up the possibility for an adaptation to reduce the overall functional lifetime of \( D \) to levels where the next scheduled *unpredictable node failure event* is beyond the functional lifetime of \( D \). Using the first *predictable node failure event* as an upper bound avoids this. Furthermore, this approach allows a direct comparison with the functional lifetime performance produced by the original SNEE infrastructure, which is inherently capped at the first *node failure event*.

The number of *unexpected node failure events* \( fnc \) is then used to determine
### Table 5.1: Average improvement between RSNEE with acquisition nodes and the other two strategies for the different query types over several unexpected node failure events.

<table>
<thead>
<tr>
<th>Type of query</th>
<th>Average improvement over static</th>
<th>Average improvement over reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>460 %</td>
<td>244 %</td>
</tr>
<tr>
<td>Aggregation</td>
<td>416 %</td>
<td>192 %</td>
</tr>
<tr>
<td>Join</td>
<td>406 %</td>
<td>200 %</td>
</tr>
</tbody>
</table>

### Table 5.2: Average improvement between the RSNEE without acquisition nodes and the other two strategies for the different query types over several unexpected node failure events.

<table>
<thead>
<tr>
<th>Type of query</th>
<th>Average improvement over static</th>
<th>Average improvement over reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>100 %</td>
<td>0 %</td>
</tr>
<tr>
<td>Aggregation</td>
<td>104 %</td>
<td>10 %</td>
</tr>
<tr>
<td>Join</td>
<td>110 %</td>
<td>25 %</td>
</tr>
</tbody>
</table>

The number of agenda cycles $q_i$ that complete before a node $fn_i$ is selected to experience the next *unexpected node failure event* $e_i$. For example, assume that $fnc = 1$ and that $q_i$ is estimated to last 80,000 agenda cycles, then a node would be selected to experience a *unexpected node failure event* after 40,000 agenda cycles. Furthermore if $fnc = 4$, then a node would be selected to experience an *unexpected node failure event* every 20,000 agenda cycles. If the strategy fails to adapt to the event $e_i$, then the estimated lifetime is taken to be the number of agenda cycles completed before $e_i$ occurred. If $fnc$ is reached, then the estimated lifetime is defined as the time taken until the strategy cannot adapt to a *predictable node failure event*. The logical flow diagram for this experiment can be found in Appendix G.4.6.

Figure 5.21(a) presents the estimated lifetimes of the topologies with a fixed level of $k_n$, for each query type, when compiled for each QoS expectations set. The following can be observed:

- In all cases, the adaptive strategies yield increased lifetimes compared to
the static strategy. This is because once the static strategy experiences the first *unpredictable node failure event*, the QEP fails, whereas both adaptive strategies repair the QEP and allows it to run for longer. Both adaptive strategies give an average improvement in lifetime compared with the static strategy, as presented in Table 5.1.

- When RSNEE is used without considering acquisition nodes in the logical overlay network, the lifetime yielded by RSNEE is comparable to the reactive strategy, as shown in Table 5.2. This is because both strategies are now limited to the lifetime of the worst physical acquisition node. This is representative of deployments where the end user does not make provision to have several acquisition nodes that are equipotent to each other. In [110] a more detailed analysis is presented of the behaviour of RSNEE and the *complete re-optimisation strategy* in deployments with a planned redundancy.

- When RSNEE is initialised to include acquisition nodes in the logical overlay network, the lifetime yielded by RSNEE is longer than that generated by the reactive strategy, as shown in Table 5.1. This is because the lifetime of \( q \) is now no longer capped at the performance of the worst individual physical node (often an acquisition node), but is now capped to the performance of the worst logical node instead.

To conclude, RSNEE outperforms the static and adaptive strategies for adapting to *unpredictable node failure events* in terms of functional lifetime, and therefore yields larger BFB values when the logical overlay network is able to support logical acquisition nodes. The extra cost of adding additional acquisition nodes should be weighed with the consideration that when this is not the case and an
(a) Estimated lifetime for all three strategies for a select * query over several unexpected node failure events for QoS set 1.

(b) Estimated lifetime for all three strategies for an aggregation query over several unexpected node failure events for QoS set 1.
acquisition node fails, the entire QEP is deemed to have failed, and this failure event could occur at any point during the QEP’s execution. When acquisition nodes are not considered in the logical overlay, RSNEE performs as well as the adaptive strategies, because they are both limited to the performance of the worst physical acquisition node because the QEP is deemed to have failed once an acquisition node has failed without a replacement.
5.5.2.2 Adaptation Energy and Time Measurements

During the adaptation process, action messages are transmitted, resulting in an extra energy drain, that has a detrimental effect on the deployment’s estimated functional lifetime. These action messages also take time, during which the deployment is not producing results. To estimate the total energy used by the adaptation process, the SNEE energy cost model [16] is invoked to estimate the cost of transmitting the adaptation messages through the RT, and the energy used in writing new binaries to flash. The logical flow diagram for this experiment can be found in Appendix G.4.2. The total time is the accumulated time used by each of the individual adaptations for each unexpected node failure event. The logical experimental flow diagram for this experiment can be found in Appendix G.4.3.

Figures 5.22 and 5.23 present the amount of energy and time used during the adaptation process for both the adaptive strategy (represented by the complete-re optimisation strategy described in Chapter 3) and RSNEE over a sequence of adaptations. The following can be observed:

- The amount of energy and time used by RSNEE in its adaptations is between 80 and 160 times smaller than is used by the complete-re optimisation strategy. This is because the adaptive strategy often requires the runtime reprogramming of a selection of nodes, whereas RSNEE does not require any binaries to be transmitted at runtime and therefore only requires the transmission of a few redirection and activation messages.

In conclusion, RSNEE generates adaptations that incur smaller energy and time overheads compared to the adaptive strategy. If adaptations can be carried out in the golden zone of an agenda cycle, the adaptations would not imply any loss of data for the end user. Because the adaptation time is shorter for RSNEE, it
is more likely that the *golden zone* will fit the adaptations, even if more frequent acquisition rates are used than are possible with the reactive strategy.

Figure 5.22: Estimated communicative energy drain upon the deployment experienced from several adaptation events for topologies with $k_n = 2$, and QoS set 1.

Figure 5.23: Estimated communicative time cost of several adaptation events for topologies with $k_n = 2$ and QoS set 1.
5.5.3 Experimental Results: Unpredictable Communication Channel Failure Events

This section investigates whether the use of RSNEE for adapting to unexpected communication channel failure events can increase the number of tuples successfully delivered to the end user by a query $q$ on a deployment $D$, where the communication channels in $D$ are susceptible to noise interference. The hypothesis is that, by increasing the number of tuples delivered, it is possible to improve the BFB value of $q$, even though the lifetime of the deployment may be adversely affected by the run-time protocol.

Note that, from now on, the assumption that communication channels are stable is removed, and therefore communication channels are now susceptible to packet loss through noise interference. Because current simulators, such as TOSSIM [73] and AVRORA [116], do not support lossy channel simulation, a well-known noise model, [70], which uses the last 20 experienced elements of a noise trace to estimate future noise levels through the use of a common pattern matching algorithm [70] was implemented. Throughout the experimental evaluation, two different noise traces were used, each of which corresponding to a different type of deployment. These are:

- A noise trace that contains relatively stable low noise levels and corresponds to a deployment in a location where there is little noise, e.g., in open fields in non-urban environments.

- A noise trace that contains erratic noise and corresponds to a deployment in a location where there are large and varying levels of noisy interference e.g., an urban environment.

When the noise model is used in conjunction with a log-distance path loss
5.5. EVALUATION

<table>
<thead>
<tr>
<th>Path loss exponent value</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6 to 1.8</td>
<td>In building line-of-sight</td>
</tr>
<tr>
<td>2</td>
<td>Free Space</td>
</tr>
<tr>
<td>2 to 3</td>
<td>Obstructed in factories</td>
</tr>
<tr>
<td>2.7 to 3.5</td>
<td>Urban area cellular radio</td>
</tr>
<tr>
<td>3 to 5</td>
<td>Shadowed urban cellular radio</td>
</tr>
<tr>
<td>4 to 6</td>
<td>Obstructed in buildings</td>
</tr>
</tbody>
</table>

Table 5.3: Path loss exponents for different environments [95]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Hardest set-up</th>
<th>Easiest set-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance between logical nodes</td>
<td>40 meters apart</td>
<td>8 meters apart</td>
</tr>
<tr>
<td>path loss exponent</td>
<td>3</td>
<td>1.6</td>
</tr>
<tr>
<td>noise trace</td>
<td>erratic noise</td>
<td>stable noise</td>
</tr>
</tbody>
</table>

Table 5.4: Different set-up parameters for the hardest and easiest set-ups.

model [95], then it is possible to simulate channels which experience unpredictable communication channel failure events through noise interference. Further explanation of the implementation of the noise model can be found in Appendix E.

The log-distance path loss model uses a path loss exponent that captures properties of different environments. Table 5.3 presents some of these path loss exponent values. During the experimental evaluation path loss exponent values of 1.6 and 3 were used to represent different environments.

There are eight different dimensions of adjustment that can be made to each experiment discussed in this section: distance between logical nodes, path loss exponent, noise trace, topology, $k_e$, $k_n$, query type and QoS expectation set. Because an exhaustive search of these combinations would result in over 9000 different experiments, the results presented in this thesis focus on just the hardest and the easiest combinations of these parameters (which are henceforth referred to as easy and hard, respectively), as presented in Table 5.4. A full exploration of the search space is presented in Appendix B.
The experiments reported in this section are divided into four separate investigations, as follows:

- The first aims to determine the extent to which RSNEE leads to an increased number of tuples being delivered to the end user over an agenda cycle. RSNEE is compared to two different strategies. These are:
  
  - RSNEE with different levels of $k_e$ and $k_n$. When RSNEE is used with $k_n > 1$ and $k_e = 1$, it is representative of RSNEE when the communication protocol is not included in the QEP agenda.
  
  - A strategy that does not adapt to unpredictable communication channel failure events, referred to as a static strategy, which has $k_e$ and $k_n = 1$. In this case, the static strategy is representative of the original SNEE infrastructure.

- The second aims to determine the extent to which the tuple delivery performance of the different strategies is affected by the distance between logical nodes.

- The third aims to determine the extent to which RSNEE leads to reduced lifetimes when a stable deployment is faced with unpredictable node failure events.

- The forth aims to determine the extent to which RSNEE leads to reduced lifetimes when an unstable deployment is faced with unpredictable node failure events and intermittent unpredictable communication channel failure events.
5.5. EVALUATION

5.5.3.1 Number of Tuples Delivered to the End User

To estimate the number of tuples delivered to the end user by a query $qu$ running on a deployment $D$, with communication channels that experience unexpected node failure events, a QEP must first be generated. This requires a physical and logical schema for $D$, an input query $qu$, and the expected $k_n$ and $k_e$. Once $q_0$ is generated, it is shipped to $D$. Fifty agenda cycles are then simulated, with each transmission task in the agenda being broken up into a number of packet transmissions, each of which is judged by the noise model as being received or dropped. The cardinality of each site task is calculated using the estimation model described in Appendix A. This is repeated until the agenda completes and the number of output tuples is recorded.

Figure 5.26 presents estimates of the number of tuples delivered to the end user per agenda cycle. In this section, only 10 different topologies, with different levels of $k_e$, for each query type is presented. A more thorough exploration of the search space can be found in Appendix B. $k_n$ does not need to be adjusted here, as $k_e$ governs how many nodes in each logical node participate in the communication protocol. The average performance improvement (measured in percentage of tuples successfully delivered to the end user) between the static strategy and RSNEE with different levels of $k_e$ for the different query types and set-ups are shown in Tables 5.5 and 5.6. The following can be observed:

<table>
<thead>
<tr>
<th>query type</th>
<th>$k_e = 2$</th>
<th>$k_e = 3$</th>
<th>$k_e = 4$</th>
<th>$k_e = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>star</td>
<td>59.4</td>
<td>59.6</td>
<td>59.6</td>
<td>59.7</td>
</tr>
<tr>
<td>aggregation</td>
<td>34</td>
<td>34.5</td>
<td>34.5</td>
<td>34.5</td>
</tr>
<tr>
<td>join</td>
<td>70</td>
<td>89.5</td>
<td>89.5</td>
<td>89.5</td>
</tr>
</tbody>
</table>

Table 5.5: Average performance increase for different levels of $k_e$ against the static strategy for the easy set-up
### Table 5.6: Average performance increase for different levels of $k_e$ against the static strategy for the hard set-up

<table>
<thead>
<tr>
<th>query type</th>
<th>$k_e = 2$</th>
<th>$k_e = 3$</th>
<th>$k_e = 4$</th>
<th>$k_e = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>star</td>
<td>21</td>
<td>39</td>
<td>49</td>
<td>59</td>
</tr>
<tr>
<td>aggregation</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>join</td>
<td>14</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
</tbody>
</table>

- Figure 5.26 (a) - (e), show that **RSNEE** outperforms the static strategy in all cases. This is because the static strategy only transmits a packet once, whereas the adaptive framework transmits the packet up to $k_e + 1$ times.

- Figure 5.26 (a) - (e), show that the performance of **RSNEE** improves as $k_e$ increases. This is because, as $k_e$ increases, the number of chances a packet has to be received increases accordingly.

- In the case of the easy set-up, **RSNEE** reaches close to perfect packet delivery with levels of $k_e$ greater than 2. This is because in environments where the noise is regularly constant and low, it is less likely that a packet will be lost on supplementary transmissions. Conversely, in the case of the hard set-up, **RSNEE** is operating at its functional limit and therefore produces less improvement.

- For the aggregation query, the count is based on the number of tuples that have contributed to the final delivered tuple. This is because an aggregation query returns only one tuple for each agenda epoch and therefore presenting the number of tuples delivered to the end user is not a useful metric for understanding the completeness of the result.

- The performance of join query is worse than both the select and aggregation query. This is because they require the input from at least two extents to
5.5. **EVALUATION**

(a) Estimated number of tuples delivered to the end user over 50 agenda cycles for select queries with different levels of $k_e$ for the hard set-up.

(b) Estimated number of tuples delivered to the end user over 50 agenda cycles for select queries with different levels of $k_e$ for the easy set-up.

Figure 5.24: Estimated number of tuples delivered to the end user over 50 agenda cycles for the select query type over different levels of $k_e$ for different set ups.
(a) Estimated number of participating tuples delivered to the end user over 50 agenda cycles for aggregation queries with different levels of $k_e$ for the hard set-up.

(b) Estimated number of participating tuples delivered to the end user over 50 agenda cycles for aggregation queries with different levels of $k_e$ for the easy set-up.

Figure 5.25: Estimated number of tuples delivered to the end user over 50 agenda cycles for a aggregation query type over different levels of $k_e$ for different set ups.
5.5. **EVALUATION**

(a) Estimated number of tuples delivered to the end user over 50 agenda cycles for join queries with different levels of $k_e$ for the hard set-up.

(b) Estimated number of tuples delivered to the end user over 50 agenda cycles for join queries with different levels of $k_e$ for the easy set-up.

Figure 5.26: Estimated number of tuples delivered to the end user over 50 agenda cycles for join query type for different levels of $k_e$ for different set ups.
generate their estimated outputs. Therefore the loss of any input packet results in a significantly higher reduction in the generated output than experienced by a select or aggregation query.

Note the poor performance of the original SNEE infrastructure, where between 20% and 40% of the expected tuples are returned to the end user. It is likely that such a performance would be considered a failure by the end user because of the loss of so many data packets. This, therefore, means that having a strategy to mitigate the effect of noise interference on the QEP’s performance is essential, for SNQPs to be considered practical in deployments where noise interference is experienced.

In conclusion, this experiment shows that RSNEE increases the performance of a QEP in terms of the percentage of tuples returned to the end user per agenda cycle and therefore could increase the BFB value for the deployment.

5.5.3.2 The Distance Effect

To explore the effect the node density has on the ability of RSNEE to successfully deliver tuples to the end user, a single topology was assessed with five different distances between nodes, as described in Section 5.5.1. The following can be observed:

- RSNEE outperforms the static strategy throughout in delivering the most tuples. This reaffirms the fact that adapting to unpredictable communication channel failure events results in improvements in tuple delivery performance.

- As $k_e$ increases, the drop in performance experienced with the increase in distance between logical nodes decreases. This is most apparent in Figures
5.5. EVALUATION

(a) Estimated percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$ on the hard set up.

(b) Estimated percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$ on the easy set up.

Figure 5.27: Estimated percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$. 
(a) Estimated percentage of tuples delivered to the end user per agenda cycle for aggregation queries over different levels of $k_e$ and $k_n$ on the hard set up.

(b) Estimated percentage of tuples delivered to the end user per agenda cycle for aggregation queries over different levels of $k_e$ and $k_n$ on the easy set up.

Figure 5.28: Estimated percentage of tuples delivered to the end user per agenda cycle for aggregation queries over different levels of $k_e$ and $k_n$. 
5.5. EVALUATION

(a) Estimated percentage of tuples delivered to the end user per agenda cycle for join queries over different levels of $k_e$ and $k_n$ on the hard set up.

(b) Estimated percentage of tuples delivered to the end user per agenda cycle for join queries over different levels of $k_e$ and $k_n$ on the easy set up.

Figure 5.29: Estimated percentage of tuples delivered to the end user per agenda cycle for different query types, levels of $k_e$ and $k_n$ and different set ups.
5.27(a) and 5.29(a), where the difference between the two furthest distances for $k_e = 5$ is 26%, whereas for $k_e = 2$ the difference is 65%. This is because, as the distance increases, the likelihood of a packet being lost due to interference increases, but the increase in $k_e$ means that there are more chances for the packet to be recovered, thereby improving the performance.

- The improvement from higher levels of $k_e$ increases as the distance between logical nodes increases. This is most apparent in Figures 5.27(b) and 5.28(b) where a level of $k_e > 2$ produces little or no improvement in the performance of RSNEE. This is because at shorter distances, the effect of the interference on packet reception is smaller and therefore fewer retransmissions are needed to ensure all packets are received, and therefore, increasing $k_e$ beyond this point results in diminishing returns.

It is worth noting that, in the case of the hardest topology set-up, the increase in $k_e$ does not result in the same degree of diminishing returns. This is because RSNEE is operating at its limit.

In conclusion, as the distance between logical nodes increases, the performance of RSNEE worsens, but this can be counteracted by increasing $k_e$. This allows an end user to make an informed decision on how to modify a deployment to achieve an expected level of performance when faced with given levels of noise interference, network density, and levels of $k_e$. Overall, this experiment showed that RSNEE does indeed increase the performance of a QEP in relation to the percentage of delivered tuples to the end user, even when the nodes are at varying distances apart, and this may increase the BFB value of the deployment.
5.5. EVALUATION

5.5.4 Robustness Measurements

In Section 3.4.2.1, each topology in a given query set was run with the same query. To show that diverse queries grouped by type do not have a significant effect on the results, a robustness analysis was carried out. An average-performing topology for each query type was selected, and then randomly generated queries (presented in Appendix D) were executed on it. Lifetimes were measured for both the adaptive and non-adaptive version of SNEE. Figure 5.30 presents the different percentage of tuple delivery per agenda cycle obtained from the various queries and Figure 5.31 presents the increase gained by RSNEE over the static strategy obtained from the various queries. The following can be observed:

- The percentage of tuples delivered per agenda cycle for a deployment can be significantly different (between 46% and 99%), depending on the query.

The conclusion from this experiment is that different queries of a particular query type have, overall, similar performance differentials between the adaptive and non-adaptive version of SNEE and, therefore, for the rest of the experimental evaluation only one query is used per query type.
Figure 5.30: The percentage of tuples delivered per agenda cycle for a given deployment over a collection of diverse queries.
Figure 5.31: The percentage increase of RSNEE over the static strategy in tuples delivered per agenda cycle for a given deployment over a collection of diverse queries.
5.5.4.1 Lifetime vs Number of Tuples in Stable Deployments

To estimate the lifetime achievable using RSNEE over a deployment $D$, first a QEP $q_i$ must be generated by inputting $qu$, physical and logical schemas into the RSNEE compilation stack. Once $q_i$ is generated, it is shipped to $D$. Then, twenty agenda cycles are simulated and the energy cost of each node over the twenty agenda cycles is averaged, to give an approximate energy cost of executing each site task for an agenda cycle. The SNEE energy cost model [16] is used to estimate the lifetime of the QEP by estimating the lifetime of each node in $D$. The node with the shortest lifetime is selected to experience a predictable node failure event $e_i$. RSNEE adapts to $e_i$, generating a new QEP which is shipped to $D$, and the next node to experience a predictable node failure event is determined. This process repeats until RSNEE cannot adapt to yet another predictable node failure event $e_{n+1}$.

Figure 5.34 presents the relationship between lifetime and the percentage of tuples delivered to the end user, per agenda cycle, for each query type, over different levels of $k_e$ and $k_n$, over both the easy and hard topology sets. The following can be observed:

- RSNEE can produce benefits of between 15% and 45% in the functional lifetime of the deployment or between 15% and 85% in the percentage of tuples delivered to the end user in relation to the static strategy representing the original SNEE architecture and occasionally both.

- Throughout the results, three distinct regions are apparent. These regions can be explained as follows:
  - One region is created by the static strategy, where $k_e$ and $k_n = 1$. This is always located near the bottom region of the field for both lifetime
5.5. EVALUATION

(a) The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$ on the hard setup.

(b) The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$ on the easy setup.

Figure 5.32: The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$. 
The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for aggregation queries over different levels of $k_e$ and $k_n$ on the hard set up.

(b) The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for aggregation queries over different levels of $k_e$ and $k_n$ on the easy set up.

Figure 5.33: The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for aggregation queries over different levels of $k_e$ and $k_n$. 
5.5. EVALUATION

(a) The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for join queries over different levels of $k_e$ and $k_n$ on the hard setup.

(b) The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for join queries over different levels of $k_e$ and $k_n$ on the easy setup.

Figure 5.34: The relationship between lifetime and the percentage of tuples delivered to the end user per agenda cycle for select queries over different levels of $k_e$ and $k_n$ for different set ups.
and tuple delivery. This indicates that having no resilience to either node or communication failure events results in poor performance in both functional lifetime and tuple delivery.

– The second region is created by the optimistic strategy, where $k_e = 1$ and $k_n > 1$. This is positioned at the far right for lifetime but also shares the bottom for tuple delivery. Overall, this indicates that if the end user caters for node failure events but not for communication failure events, then the user should expect between 5% and 60% of the tuples to be delivered per agenda cycle, but with a boost in overall functional lifetime. This is most apparent in Figures 5.32(a) and 5.32(b) where the static strategy results in an lifetime of 57,000 agenda cycles, and the optimistic strategy ($k_e = 1$ and $k_n > 1$) results in an lifetime of between 92,000 to 98,000 agenda cycles, an increase of between 61% and 72% therefore.

– The third region is created by RSNEE, where $k_e > 2$. This is always located above the other two regions and represents a different level of performance. Overall, this indicates that if the user caters for both types of failure events, then the overall performance in tuple delivery can be increased, but, in doing so, the user relinquishes a certain amount of functional lifetime. This is most apparent in Figures 5.34(a) and 5.34(b), where the lifetime performance of the optimistic strategy ($k_e = 1$ and $k_n > 1$) is 80,000 agenda cycles, whereas the performance of the pessimistic strategy ($k_e = k_n$) is 18,000 agenda cycles.

It is worth noting that because the optimistic strategy results in 2% of the expected tuples being delivered to the end user, and the pessimistic one returns 50% of the expected tuples, the pessimistic strategy returns, overall, more of the
tuples. For example, assume 100 tuples were expected to be delivered in each agenda cycle, then the optimistic strategy would return a total of 160,000 tuples whereas the pessimistic one would return 900,000 tuples, an increase of 462% in performance.

A more thorough presentation of the results for all the different combinations of difficulty (noise trace, distance, path loss exponent) can be found in Appendix B. It is worth noting that the performance of RSNEE in respect of lifetime with the same query type is approximately the same, with a variance of 0.02%, for all possible combinations of set-up parameters. This is because RSNEE rarely ensures that all 100% of the tuples are delivered to the end user and therefore it is always executing all of the redundant tasks, and so seldom goes into energy saving mode.

This trade-off between lifetime and percentage of tuples delivered delegates to end users the need to decide on this priority. If the end user wishes to have the deployment last as long as possible, with less consideration as to a gapless sequence of results, then the optimistic strategy should be the chosen set up. If the end user is intolerant of gaps in the results even if the deployment thereby fails to last as long as it might, then the pessimistic strategy should be the chosen set-up.

Figure 5.35 presents the total number of tuples delivered to the end user over the deployment’s functional lifetime for the hard topology set-up. The following can be observed:

- If the end user wanted the best of both worlds and can afford to deploy RSNEE with a resilience level of $k_n = 5$, then $k_e = 3$ returns the largest number of tuples.
- For lower levels of $k_n$, the optimal value for $k_e$ varies between 2 and 3,
Figure 5.35: The number of tuples delivered to the end user during the lifetime of the deployment for different levels of $k_e$ and $k_n$.

implying that a focus on improving the number of packets delivered to the end user is more beneficial than trying to increase the functional lifetime of the deployment to increase the BFB value.

In conclusion, there is a detrimental effect in using RSNEE with the communication protocol in terms of functional lifetime. Yet this detrimental effect can be outweighed by the level of $k_e$, thereby still ensuring an increase in the overall BFB value.
5.5.4.2 Lifetime in Unstable Deployments

To estimate the lifetime achievable using RSNEE over a deployment $D$, which can experience *unpredictable node failure events*, first a QEP $q_i$ must be generated by inputting $qu$, physical and logical schemas into the RSNEE compilation stack. Once $q_i$ is generated, it is shipped to $D$. Then 20 agenda cycles are simulated and the energy cost of each node over the 20 agenda cycles is averaged, to give an approximate energy cost of executing each *site task* for an agenda cycle. The SNEE energy cost model [16], is used to estimate the lifetime of the QEP by estimating the lifetime of each node in the QEP. The shortest lifetime among nodes is selected as the time during which *unpredictable node failure events* can occur. The logical flow diagram for this experiment can be found in Appendix G.4.6. In this case, any node can be selected to experience a *unpredictable node failure event*, because the logical overlay network is initialised with acquisition nodes being considered for logical node status.

It was previously noted that the lifetime variance for stable deployments, with different difficulty levels in their set-ups, was 0.02%. This means that, for the unstable deployment experiments, a single set of parameters is sufficient to realistically represent all combinations of difficulty. The chosen one was the hard set-up.

Figure 5.38 presents the lifetime of the strategies when deployed on a deployment that has experienced a number of *unpredictable node failure events* for each query type. The following can be observed:

- As the number of *unexpected node failure events* increases, the lifetime of the deployment often only changes by an average of 5%. This is mainly because, whereas in previous experiments there were $x$ nodes participating in the QEP, there are now $k_n \times x$ nodes, each of which can be selected to
fail, and therefore, the chance that one of the physical nodes selected to experience the \textit{unpredictable node failure events} participates in the logical node that is the first to completely fail are reduced to one in $k_n \times x$. This means that the \textit{unpredictable node failure events} have only minimal effects on the lifetime, as the only energy cost incurred is by the adaptation, which, as shown in Section 5.5.2.2, is little in relation to the execution cost of the QEP over many agenda cycles.

- The majority of the changes in lifetime can be accounted for by the variances in each site task’s execution cost between adaptations. These variances are directly related to the increased cost of tuple transmission between replacement nodes and their children, which over the lifetime of their logical nodes, can result in lifetime variances measuring up to 10,000 agenda cycles.

- When a physical node that participates in the logical node that fails first is selected to experience an \textit{unpredictable node failure event}, the lifetime of the deployment drops accordingly. This is most apparent in Figure 5.36(b) with $k_e = 2$ and $k_n > k_e$ for 5, 9 and 16 \textit{unpredictable node failure events}, where the lifetime drops suddenly.

To consider how this behaviour affects the BFB value for a query, consider the query $Q$ represented in Figure 4.4(a). The QEP generated from $Q$ is assumed to generate 495 tuples per agenda cycle. Now assume that the QEP $q_s$ has been generated by the original SNEE infrastructure for $Q$. Now assume that $q_s$ is executing on the deployment $D$ which can experience both \textit{unpredictable node and communication channel failures} and costs £10000 to deploy. Now let's assume that $D$ experiences a \textit{unpredictable node failure} after approximately 30000 agenda cycles (approximately half its expected functional lifetime from energy depletion alone) and regularly experiences a low level of interference throughout its execution and
5.5. Evaluation

(a) The lifetime of the strategies when deployed on a deployment with select query where $k_n = 3$.

(b) The lifetime of the strategies when deployed on a deployment with select query where $k_n = 5$.

Figure 5.36: The lifetime of the strategies when deployed on a deployment with select query
(a) The lifetime of the strategies when deployed on a deployment with aggregation query where $k_n = 3$.

(b) The lifetime of the strategies when deployed on a deployment with aggregation query where $k_n = 5$.

Figure 5.37: The lifetime of the strategies when deployed on a deployment with aggregation query
5.5. EVALUATION

(a) The lifetime of the strategies when deployed on a deployment with join query where $k_n = 3$.

(b) The lifetime of the strategies when deployed on a deployment with join query where $k_n = 5$.

Figure 5.38: The lifetime of the strategies when deployed on deployments which have experienced a number of unpredictable node failure events.
therefore only delivers 40% of its expected tuples. At this point, \( q_s \) is considered to have failed and produces a BFB of \( \frac{10000}{(495 \times 30000 \times 0.4)} \) or £0.0016 per tuple. Now consider a QEP \( q_{rs} \) that has been generated by the RSNEE infrastructure for \( Q \) with \( k_e \) set to 2 and is executed on a recharged \( D \) which has been modified to support a \( k_n \) resilience of 2 and therefore costs £20000 to deploy. Assume that \( D \) experiences the same unpredictable node failure at agenda cycle 30000 as well. In this case, \( q_{rs} \) adapts and carries on executing for another 40000 agenda cycles before failing completely and during each agenda cycle, delivered 99% of the expected tuples. The BFB value for \( q_{rs} \) is \( \frac{20000}{(495 \times 70000 \times 0.99)} \) or £0.000583 per tuple which represents a 288% increase in BFB value.

In conclusion, the detrimental effect of unpredictable node failure events on the performance of RSNEE is minimal most of the time, but when an unpredictable node failure event is experienced by the weakest logical node then a drop in functional lifetime is observed. Even though the deployment has been designed to be resilient to a minimum of \( k_n \) node failure events, it should be possible to increase the functional lifetime by redistributing these redundant nodes to support non-uniform levels of \( k_n \), with a focus on logical-node hot-spots. This is left for future work as discussed in Section 6.3.4. It is worth noting that by being resilient to node failure, the BFB value for QEPs generated by the RSNEE infrastructure is significantly increased over what is possible by the original SNEE infrastructure.

5.6 Related Work

As discussed in Section 1.3, current SNQPs have little or no adaptive behaviour integrated into or around their running QEPs. In the context of unpredictable
communication channel failure events, a few SNQPs have limited support. Neither TinyDB [80] nor the original SNEE SNQP [32] consider handling intermittent unpredictable channel failure events. TinyDB [80] exhibits a degree of adaptive behaviour in terms of permanent unpredictable channel failure events through its semantic routing tree, since a node $n$ can switch its parent from $p$ to $p'$ if $n$ has not heard from $p$ within a given time period. Because (unlike SNEE) TinyDB sends the entire QEP to every node in the deployment, switching parents is a sound strategy.

SmartCIS [77] and AnduIN [66] both use a simple form of TCP/IP to handle both types of unpredictable channel failure events. TCP/IP uses acknowledgement packets in conjunction with packet retransmissions to support reliable communication channels. AnduIN gets its TCP/IP protocol stack from the Contiki operating system. Both of these techniques mitigate unpredictable channel failure events, but without considering the amount of time it takes to do so. Therefore the use of TCP/IP is found to have a detrimental effect on a QEP’s ability to meet delivery time expectations.

The most relevant case of related work is in adaptive MAC protocols. These cannot be used in the SNEE architecture, since the choice is for a globally-timed agenda to handle the execution of both communication transmissions and in-network computations. It is likely that introducing such protocols to a SNEE QEP would result in a QEP that exhibits a worse performance in terms of meeting certain QoS expectations such as delivery time.

5.6.1 Predicting Communication Channel Failure Events

Apart from adaptive protocols, there are a few proposals that aim to determine which channels in a deployment are likely to experience channel failure events.
and to counteract them [68, 30, 104, 129, 39].

These proposals determine which channels are likely to experience *channel failure events*, based on metrics such as received signal strength indicator (RSSI) values or signal to noise ratios (SNR) [68, 30, 129]. If the channel’s metric drops below a given threshold, they are deemed to be at risk of experiencing *channel failure events* and are blacklisted from RT constructions. These proposals do not consider, or adapt to, situations where a previously determined, reliable channel, experiences an *unpredictable communication channel failure event*, and therefore becomes unreliable. This means that such proposals can only be effective when used in conjunction with an adaptive runtime strategy.

In hierarchical deployments, where clustering proposals are often used, avoiding channels deemed unreliable is done in the cluster-formation stage [39]: each node connects to the cluster head which has the best SNR. Hierarchical structures, as mentioned in previous chapters, execute little or no in-network computation, and are likely to be less energy-efficient than RSNEE discussed in this chapter, due to overheads incurred through repeated cluster formation and reprogramming.

### 5.6.2 Reacting to Communication Channel Failure Events

Other proposals in the area are reactive in nature, and their behaviours can be broken down into three distinct types.

The first behaviour can be categorised as overhearing. In this case, other nodes, such as \( n' \), are programmed to overhear the transmission of \( n \), and if \( n' \) determines that the parent of \( n \) has not received \( n \)'s packet, \( n' \) retransmits them [90]. This behaviour closely resembles the RSNEE’s runtime communication protocol, except that it is limited to exactly one overhearing node and does not consider *unexpected node failure events* or the effect of the re-transmission on
other QEP operations.

The second behaviour can be categorised as adapting transmission power. In this case, the transmission power used by the nodes is adjusted up or down in the hope of reducing the chances of experiencing future *unpredictable communication channel failure events*, while also being energy conscious [75, 112]. These proposals do not try to recover from the current *unpredictable communication channel failure event*, but try to avoid experiencing them in the future. Boosting the transmission power of the transmitting node can only have a limited effect on avoiding future *unpredictable communication channel failure events*, as the main source of these events is environmental noise, which can vary in its intensity.

Finally, some proposals attempt to carry on with using the unreliable channel, regardless of whether the channel has experienced a permanent *unpredictable communication channel failure event* or a temporary one [69, 27, 125]. These proposals use retransmissions, in the hope that eventually the channel will become reliable again at some point in the near future. Because these proposals do not consider the amount of time used up on a hop-by-hop basis, it is possible for all the packets to fail to reach the end user in the specified QoS delivery time, resulting in the QEP being considered a failure, from the end user’s perspective.

### 5.7 Conclusions

In this chapter, a strategy for adapting to the two types of *unpredictable failure events* has been presented in the form of RSNEE. RSNEE can be broken down into two different aspects, each of which is focused on dealing with one type of *unpredictable failure event* and has been described with the use of a working example. The main conclusion gained from the evaluation for *unpredictable node failure events* is that when operating on resiliently designed deployments, RSNEE
returns significantly higher functional lifetimes compared with a non-adaptive strategy or the reactive strategies described in Chapter 3, and therefore produces a better BFB value than not adapting to unpredictable node failure events.

The main conclusion gained from the evaluation for unpredictable communication channel failure events is that RSNEE returns a significantly higher percentage of result tuples to the end user in comparison to an non-adaptive strategy. When taken into account with the trade-off in functional lifetime, the BFB value can still be increased beyond what is capable without the technique.

The next chapter presents a review of the research contributions made in this thesis and proposes future research directions that arise from these results.
Chapter 6

Conclusions

In Section 6.1 of this chapter a review of the research contributions made in this thesis is presented. Section 6.2 discusses the significance of the results obtained in this thesis and Section 6.3 presents proposed future work directions that arise from these results.

6.1 Research Contributions

This section summarises the research contributions made in this thesis in relation to the research objectives defined in Section 1.4.

Two distinct contributions are associated with objective O1 (adapting to unpredictable node failure events). The first contribution involved the design, implementation and evaluation of two techniques that mitigate unpredictable node failure events in SNQP QEPs. These QEPs are executed on deployments where there is little or no planned node redundancy. These strategies generate functionally equivalent QEPs that avoid using the node that experienced the unpredictable failure events. This resulted in an instantiation of SNEE, referred to as ADSNEE, described in Section 3.1.
The second contribution associated with objective O1 is a strategy that operates on deployments that have been deliberately designed with extra redundancy. This strategy mitigates *unpredictable failure events* through the use of a logical overlay abstraction that is computed at compile time, as described in Chapter 5. This strategy resulted in the RSNEE instantiation of SNEE which supports the resilience to *k node failure events*, where *k* is the number of nodes that participate in a logical node.

Adapting to *unpredictable node failure events* when they occur is beneficial, as it supports longer lifetimes for the executing query than can be achieved by not adapting to such failures. This, in turn, enables a larger BFB metric for the deployment, as demonstrated by the experimental evidence in Section 3.4. Being able to generate QEPs that result in small levels of runtime reprogramming when switching between the currently-executing QEP, and the new QEP, leads to quicker adaptations. This can, in turn, result in adaptations that can be executed during the *golden zone* of the QEP agenda. This results in adaptations that seem to be seamless to the end-user as there is no interruption in the stream of results, as demonstrated by the experimental evidence in Section 3.4.2.4.

Objective O2 involved the design, implementation and evaluation of techniques that mitigate *predictable node failure events* in SNQP QEPs. The resulting research contribution was a strategy that planned a sequence of plan transitions at compile time that avoid energy depletion and can therefore be followed until an *unpredictable node failure event* occurs, as described in Chapter 4. This proactive strategy resulted in the ADPSNEE instantiation of SNEE as described in Section 4.1. This strategy therefore provides resilience to both *predictable* and *unpredictable node failure events*. Planning adaptations ahead of time is beneficial, as it supports longer lifetimes for the query than can be achieved with a purely reactive strategy. This in turn results in larger BFB values for the deployment,
as demonstrated by the experimental evidence in Section 4.3.2. The ADSNEE instantiation, therefore, makes WSN deployments with no intentionally planned redundancy more cost effective in environments where node failure events can occur.

The contribution associated with Objective O3 involved extending the logical overlay strategy, described in Chapter 5, with a simple runtime protocol that enables the communication between logical nodes to exploit the $k$ links available, as described in Section 5.4. This strategy results in a version of SNEE, referred to as RESNEE, which supports the resilience to both types of unpredictable failure events covered in this thesis. To support this level of resilience, a decision must be made over either focusing on extending the functional lifetime of the deployment or increasing the percentage of the expected tuples that are successfully delivered to the end user at the end of each agenda cycle. By focusing on either goal, an end user can increase the BFB value, as demonstrated by the experimental evaluation in Section 5.5.

### 6.2 Significance of the Results

As mentioned several times, the QEPs generated by the different SNQPs in the literature, exhibit little or no resilient behaviour in the face of different types of failure events that can occur in a WSN. This is mainly because the focus of these SNQPs is to generate energy-efficient QEPs that try to meet diverse types of QoS expectations demanded of the application by end users.

Resilience to different types of failure events is an important concern, as demonstrated by the amount of research carried out in recent years from the WSN community, targeted at mitigating or adapting to these events when they occur. Yet, in the SNQP community, where failure events can have severe effects
on the executing QEP, much less work has been done, as demonstrated by research proposals that take a pre-existing SNQP, such as TinyDB, and extend it to handle *unexpected node failure events* [77].

The research contributions in this thesis demonstrate that SNQP-generated QEPs can be made resilient to both *predictable and unpredictable node and communication channel failure events*, whilst still ensuring the QEPs meet the QoS expectations placed by the end user. In doing so, SNQPs will become more general purpose and cost-effective in environments where both *predictable and unpredictable node and communication channel failure events* can occur. This is significant, given that human intervention is often required to repair WSNs e.g., by sending an person to replace batteries in motes. This is undesirable wherever deployments are located in hard to reach or hostile environments.

Another significant advantage of using a resilient form of SNQP is that such SNQPs support lifetime increases beyond what is possible with non-resilient forms, thereby increasing BFB values and the cost-effectiveness of using SNQP technology on WSNs.

It is worth noting that the research contributions in this thesis were implemented as an extension to the SNEE [32] SNQP. These same contributions would not work in SNQPs where there is no precomputed routing tree, typically because of the use of a routing protocol at runtime. Examples of such SNQPs are TinyDB, Smartcis and AnduIN [80, 77, 66].

### 6.3 Future Work Directions

The contributions made in this thesis suggest several directions for future work. Broadly speaking, these involve either extending the resilient infrastructure described in this thesis to take into account other aspects of resilience, or expanding
6.3. FUTURE WORK DIRECTIONS

the current infrastructure so that it can operate over different forms of WSNs.

6.3.1 Resilience Aspects

- **Clock Drift.** With the current infrastructure, it is assumed that the motes in the deployment have their clocks synchronised at all times. In real deployments, this is possible initially, as explained in many proposals (e.g., [102, 55, 100, 105]), but as these deployments are expected to operate for long periods of time without human intervention, there is ample opportunity for the motes to get out of sync with each other. This is a problem for SNEE-generated QEPs, as the agenda associated with an executing QEP is generated at compile time, and regulates when both nodes that participate in a communication task should activate their radios. This means that clock drift could result in the two nodes being active at different times and missing each others’ transmissions. Thus, the incorporation of support for detecting when a node clock has drifted and the correcting it before it becomes a problem for the executing QEP is likely to be useful future research. Ideally, this should be possible using the current transmission of tuple packets and without disrupting the strategies proposed in this thesis. Exploiting the acknowledgement packet transmitted by the receiver in the logical overlay strategy should suffice.

- **Node Failure Detection in Noisy Environments.** With the research contributions discussed in this thesis there is an assumption that it is possible to detect from the tuples returned when a node has failed (see Section 3.1.1). In noisy environments, it may not be possible to ensure that 100% of the relevant tuples reach the base station, and this would in turn cause erroneous failure detections. Incorporating a detection mechanism that can
cope with the possibility of losing a fraction of the tuples that should be delivered to the base station would improve the performance of the strategies discussed in this thesis in noisy environments. Again, this should be possible with no modification to the strategies proposed in this thesis.

6.3.2 Other Types of Deployments

- **Multiple Base stations.** All deployments considered in this thesis have one base station. If there were more base stations in the deployment, there would be the potential to generate QEPs that transmit their data to different base stations, and therefore distribute the workload over more nodes in the deployment. This would also make the operating QEPs more resilient, as it would remove the assumption that the base station is powered by mains power supply and remove the single point of failure that the base station is, in all deployments considered in this thesis. This is not currently possible with any instantiation of SN EE and would likely result in complete rewrites of the algorithms in the multi-site phase of the original SN EE compilation stack, as well as modifications to some of the strategies presented in this thesis.

- **Multiple Executing QEPs.** All the experiments presented in this thesis assume that only one query is running in a given deployment during its functional lifetime. In real-world deployments, it is often the case that the deployment will be re-purposed on-the-fly to account for changing end-user requirements. As WSN, become more main stream, it is probable that a deployment will be used by multiple end users at the same time, for different purposes, with potentially contradictory QoS expectations. Therefore, implementing the ability to execute multiple QEPs in the same deployment
would be a useful line of exploration and research. Ideally, this should be possible, with modifications to the multi-site phase of the original SN EE compilation stack.

6.3.3 Distributed Proposal

- **Distributed Proposal.** All of the proposals presented in this thesis execute some or all of their processing in a centralised fashion (by operating on global information outside the deployment). It would be an interesting avenue of research to explore if it is possible to generate RS NEE -like or AD-S NEE -like QEPs that ensure they meet QoS expectations without global information or a centralised computation point. This would require a complete re-writes of the entire infrastructure of the centralised algorithms into distributed ones. Yet having a distributed form of either RS NEE or ADS NEE would potentially result in faster adaptations, due to removing the need to relay information out of the deployment to determine when a failure event occurs.

6.3.4 Improvements to Techniques

- **Multiple iterations of a logical communication edge.** The logical communication edge contains one iteration of attempting to re-transmit packets via the nodes within the logical node. If the agenda contains enough spare time to generate multiple iterations of a logical edge, it may improve packet delivery rate whilst saving energy via the redundancy tasks.

- **Non-uniform distribution of physical nodes in the logical node.** The non uniform energy depletion of logical nodes result in logical nodes
that have higher workloads failing earlier than nodes with smaller workloads. Distributing the nodes so that high work load logical nodes contain more physical nodes, could increase the lifetime of the deployment and therefore the BFB metric.

The above proposals for future work indicate that the further exploration of query processing in WSN has the potential to improve the practicality of WSN’s as well as yield significant new research results.
Appendix A

Cardinality Model:
Supplementary Material

This appendix can be located in the appendix folder on the disk under the file name cardinalityModel.pdf
Appendix B

Cardinality Model Evaluation

This appendix can be located in the appendix folder on the disk under the file name cardinalityEval.pdf
Appendix C

Single Node Failure:
Supplementary Material

This appendix can be located in the appendix folder on the disk under the file name unpredictableLifetimeSingle.pdf
Appendix D

Robustness Measurements:
Supplementary Material

This appendix can be located in the appendix folder on the disk under the file name robustness.pdf
Appendix E

Noise Model: Supplementary Material

This appendix can be located in the appendix folder on the disk under the file name noise.pdf
Appendix F

SNEEq\textit{l} Algebra: Supplementary Material

This appendix can be located in the appendix folder on the disk under the file name sneeq\textit{l}.pdf
Appendix G

Experimental Flows:

Supplementary Material

This appendix can be located in the appendix folder on the disk under the file name experimentalFlows.pdf
Appendix H

Experimental Edge lifetime to Tuple relationship:

Supplementary Material

The experimental results can be found with the supplement disk.
Bibliography


[77] M. Liu, S. R. Mihaylov, Z. Bao, M. Jacob, Z. G. Ives, B. T. Loo, and


the 3rd international conference on Embedded networked sensor systems, SenSys ’05, pages 76–89, New York, NY, USA, 2005. ACM.


