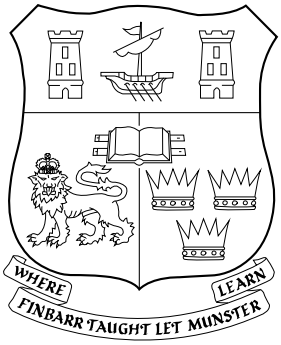
Planning the Deployment of Fault-Tolerant Wireless Sensor Networks

Lanny Sitanayah



A Thesis Submitted to the National University of Ireland

in Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Computer Science

January 2013

Research Supervisors: Prof. Cormac J. Sreenan
Dr. Kenneth N. Brown
Head of Department: Prof. Barry O'Sullivan

Department of Computer Science
National University of Ireland, Cork

Declaration

This thesis is submitted to University College Cork, in accordance with the requirements for the degree of Doctor of Philosophy in Computer Science. The research presented in this thesis are entirely my own work and have not been submitted to any other university or higher education institution, or for any other academic award in this university. Where use has been made of other people's work, it has been fully acknowledged and referenced.

Lanny Sitanayah

January 2013

Abstract

Wireless Sensor Networks (WSNs) are comprised of many low-cost sensor nodes that communicate using wireless links. Since WSNs are subject to failures, fault-tolerance becomes an important requirement for many WSN applications where messages from all sensor nodes must be delivered to data sinks in a reliable and timely manner. Fault-tolerance can be enabled in a number of different areas of WSN design and operation, including the Medium Access Control (MAC) layer and the initial topology design. We show that MAC protocols and topology planning algorithms can be designed together to create fault-tolerant WSNs for volatile environments.

To be robust to failures, a MAC protocol must be able to adapt to sudden traffic fluctuations and topology dynamics, for which we design ER-MAC, a hybrid MAC protocol for emergency response WSNs. ER-MAC is able to switch from energy-efficient operation in normal periodic monitoring to reliable and fast delivery for emergency monitoring, and vice versa. It also has a special functionality to prioritise high priority packets and guarantee fair packet deliveries from all sensor nodes.

Topology design can support fault-tolerance by ensuring that there are alternative acceptable routes to data sinks when failures occur. We provide solutions for four topology planning problems: Additional Relay Placement (ARP), Additional Backup Placement (ABP), Multiple Sink Placement (MSP), and Multiple Sink and Relay Placement (MSRP). Our solutions use a local search technique based on Greedy Randomized Adaptive Search Procedures (GRASP).

GRASP-ARP deploys relays for (k, l)-sink-connectivity, where each sensor node must have k vertex-disjoint paths of length $\leq l$. To count how many disjoint paths a node has, we propose Counting-Paths and its dynamic programming variant. While GRASP-ARP ensures the length-bound with the basic Counting-Paths, it runs faster with the dynamic programming variant. GRASP-ABP deploys fewer relays than GRASP-ARP by focusing only on the most important nodes – those whose failure has the worst effect. To identify the most important nodes, we define a new centrality measure, Length-constrained Connectivity and Rerouting Centrality (l-CRC).

For the MSP and MSRP problems, besides presenting GRASP-MSP and GRASP-MSRP, we also develop greedy algorithms, which we called Greedy-MSP and Greedy-MSRP. Greedy-MSP and GRASP-MSP place multiple sinks with minimal cost to ensure that each sensor node in the network is double-covered, i.e. has at least two length-bounded paths to two sinks. Greedy-MSRP and GRASP-MSRP deploy multiple sinks and relays with minimal cost to make the network double-covered and non-critical. Non-critical means all sensor nodes must have length-bounded alternative paths to sinks when an arbitrary sensor node fails.

We evaluate the fault-tolerance of each deployment result in multi-hop data gathering simulations using ER-MAC. In simulation, the topologies of GRASP-ARP and GRASP-ABP show comparable performance, even though GRASP-ABP requires fewer relays than GRASP-ARP. For the multiple sink scenario, the topologies of GRASP-MSRP achieve the best performance because of better sink positions.

Acknowledgements

Praise be the Lord because of His blessing and guidance, I am able to present this thesis on the completion of my PhD study. At the end of this 3 years and 10 months journey, I would like to thank many people who has helped and supported me.

I would like to thank my supervisors Prof. Cormac Sreenan and Dr. Ken Brown for their precious time given in supervising, valuable advice, help and never-ending support. I really appreciate their effort to read my drafts and make corrections, without which this thesis would have not been possible.

I also wish to thank my thesis examiners: Prof. Silvia Santini from Technische Universität Darmstadt and Dr. Steve Prestwich from University College Cork for the priceless suggestions and recommendations for the thesis revision.

I wish to thank NEMBES and CTVR fellows: Dr. Tatiana Tabirca, Dr. Yuanyuan Zeng, Dr. Xiuchao Wu, Dr. Tarik Hadzic, Dr. David Stynes, and Seán Óg Murphy for fruitful discussions and suggestions for my research. I would like to express my special thanks to Ms. Mary Noonan for taking care of administrative matters. I would also like to acknowledge Tony O'Donovan for his friendship and help. May he rest in peace.

My study is fully funded by the NEMBES project, supported by the Irish Higher Education Authority PRTLI-IV research program.

The last but not the least, I wish to thank my mother, my sister and my husband for their understanding and continuous emotional support through the difficult times.

Publication

Some of this work has already been published in peer-reviewed publications. The bibliographical details of the works and where they appear in this thesis are outlined below.

[1] L. Sitanayah, K. N. Brown, and C. J. Sreenan. Multiple Sink and Relay Placement in Wireless Sensor Networks. In *Proc. 1st Workshop Artificial Intelligence for Telecommunications and Sensor Networks (WAITS'12), 20th European Conf. Artificial Intelligence (ECAI'12)*, pages 18–23, Aug. 2012.
This paper forms Chapter 7 of this thesis.

[2] L. Sitanayah, K. N. Brown, and C. J. Sreenan. Fault-Tolerant Relay Deployment Based on Length-Constrained Connectivity and Rerouting Centrality in Wireless Sensor Networks. In G. P. Picco and W. Heinzelman (editors), *Proc. 9th European Conf. Wireless Sensor Networks (EWSN'12)*, Volume 7158 LNCS, pages 115–130, Feb. 2012.
This paper forms Chapter 6 of this thesis.

[3] L. Sitanayah, K. N. Brown, and C. J. Sreenan. Fault-Tolerant Relay Deployment for k Node-Disjoint Paths in Wireless Sensor Networks. In *Proc. 4th Int'l Conf. IFIP Wireless Days (WD'11)*, pages 1–6, Oct. 2011.
This paper forms Chapter 5 of this thesis.

[4] L. Sitanayah, C. J. Sreenan, and K. N. Brown. ER-MAC: A Hybrid MAC Protocol for Emergency Response Wireless Sensor Networks. In *Proc. 4th Int'l Conf. Sensor Technologies and Applications (SENSORCOMM'10)*, pages 244–249, Jul. 2010.
This paper forms Chapter 4 of this thesis.

[5] L. Sitanayah, C. J. Sreenan, and K. N. Brown. Poster Abstract: Emergency Response MAC Protocol (ER-MAC) for Wireless Sensor Networks. In *Proc. 9th ACM/IEEE Int'l Conf. Information Processing in Sensor Networks (IPSN'10)*, pages 364–365, Apr. 2010.
This paper contains the preliminary findings of the work in [4].

My contributions in these publications were at least 85%. I developed, implemented, and simulated the proposed solutions and wrote the papers. My supervisors, Prof. Cormac Sreenan and Dr. Ken Brown, reviewed the papers and provided useful feedback to improve the quality and readability of the papers.

List of Acronyms

1tFTP	Single-tiered Fault-Tolerant Relay Placement
1tRNP	Single-tiered Relay Node Placement
1tTSP	Single-tiered Traveling Salesman
2CRNDC	2-Connected Relay Node Double Cover
2tFTP	Two-tiered Fault-Tolerant Relay Placement
2tRNP	Two-tiered Relay Node Placement
2tTSP	Two-tiered Traveling Salesman
ABP	Additional Backup Placement
ACK	Acknowledgement
APC	Alternative Path Centrality
ARP	Additional Relay Placement
B-MAC	Berkeley-MAC
BSL	Find the Best Sink Location
BSLFT	Find the Best Sink Locations for Fault-Tolerance
CBS	Cluster-Based Sampling
CBS-BSL	Cluster-Based Sampling for Finding the Best Sink Locations
CBS-MSP	Cluster-Based Sampling for Multiple Sink Placement
CBS-MSRP	Cluster-Based Sampling for Multiple Sink and Relay Placement
CCA	Clear Channel Assessment
COLA	Coverage and Latency Aware Actor Placement
CRNSC	Connected Relay Node Single Cover
CSMA	Carrier Sense Multiple Access

CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CT	Connectivity Centrality Threshold
CTL	Control
CTP	Collection Tree Protocol
CTS	Clear-To-Send
DECOMP	Iterative Decomposition
DP	Dynamic Programming
DSSS	Direct-Sequence Spread-Spectrum
EB-MAC	Event Based MAC
ER-MAC	Emergency Response MAC
ECN	Explicit Congestion Notification
FDMA	Frequency-Division Multiple Access
FLAMA	FLow-Aware Medium Access
FTSP	Flooding Time Synchronisation Protocol
GADO	Genetic Algorithm for Distance Optimisation
GAHO	Genetic Algorithm for Hop Count Optimisation
GRASP	Greedy Randomised Adaptive Search Procedure
GRASP-ABP	Greedy Randomised Adaptive Search Procedure for Additional Backup Placement
GRASP-ARP	Greedy Randomised Adaptive Search Procedure for Additional Relay Placement
GRASP-MRP	Greedy Randomised Adaptive Search Procedure for Multiple Relay Placement
GRASP-MSP	Greedy Randomised Adaptive Search Procedure for Multiple Sink Placement
GRASP-MSRP	Greedy Randomised Adaptive Search Procedure for Multiple Sink and Relay Placement
Greedy-MSP	Greedy Algorithm for Multiple Sink Placement
Greedy-MSRP	Greedy Algorithm for Multiple Sink and Relay Placement

GRASP-SPG	Greedy Randomised Adaptive Search Procedure for Steiner Tree Problem in Graphs
HCL	High Contention Level
HOMP	Heuristic Opt Multisink Place
K**-CONN-REPAIR**	The Partial k-Connectivity-Repair Algorithm
l**-CC**	Length-constrained Connectivity Centrality
l**-CRC**	Length-constrained Connectivity and Rerouting Centrality
l**-RC**	Length-constrained Rerouting Centrality
LAND	Localised Algorithm for Finding Node-Disjoint Paths
LCL	Low Contention Level
LPL	Low Power Listening
MAC	Medium Access Control
MBCP	Maximally Balanced Connected Partition
MINLP	Mixed Integer Non-linear Programming
MRP	Multiple Relay Placement
MRP-1	Minimum Relay-Node Placement for 1-Connectivity
MRP-2	Minimum Relay-Node Placement for 2-Connectivity
MSFT	Minimise the Number of Sinks for Fault-Tolerance
MSP	Multiple Sink Placement
MSPOP	Minimise the Number of Sinks for a Predefined Minimum Operation Period
MSRFT	Minimise the Number of Sinks and Relays for Fault-Tolerance
MSRP	Multiple Sink and Relay Placement
MTWP	Multi-hop Traffic-flow Weight Placement
PEQ	Periodic, Event-driven and Query-based routing protocol
PMAC	Pattern-MAC
PMP	P-Median Problem

QoB	Quality of Backup
QoS	Quality of Service
RNP$_{\mathrm{C}}$	Connected Relay Node Placement
RNP$_{\mathrm{S}}$	Survivable Relay Node Placement
RRMAC	Real time and Reliable MAC
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
RT	Rerouting Centrality Threshold
RTS	Request-To-Send
S-MAC	Sensor-MAC
SPG	Steiner Tree Problem in Graphs
STR	Shortest Path Tree Routing
T-MAC	Timeout-MAC
TA-MAC	Traffic Adaptive-MAC
TDMA	Time-Division Multiple Access
TPSN	Timing-sync Protocol for Sensor Networks
TRAMA	TRaffic Adaptive Medium Access
TSP	Traveling Salesman Problem
UDP	User Datagram Protocol
VTS	Virtual TDMA for Sensors
WiseMAC	Wireless Sensor MAC
WSN	Wireless Sensor Network
Z-MAC	Zebra-MAC

Contents

Declaration iii

Abstract v

Acknowledgements vii

Publication ix

List of Acronyms xi

1 Introduction 1

1.1 Objectives 3

1.2 Thesis Contributions 4

1.2.1 Thesis Statement 4

1.2.2 Proposed Solutions 4

1.3 Thesis Overview 6

2 Literature Review 7

2.1 Introduction 7

2.2 WSN Communication Architecture 7

2.3 MAC Protocols for WSN 9

2.3.1 Contention-based MAC Protocols 13
2.3.2 Schedule-based MAC Protocols 23
2.3.3 Hybrid MAC Protocols 27
2.3.4 Discussion 36
2.4 Relay Placement Algorithms 37
2.4.1 Single-tiered Relay Placement Problem 41
2.4.2 Two-tiered Relay Placement Problem 47
2.4.3 Discussion 50
2.5 Disjoint Path Algorithms 51
2.6 Centrality and Alternative Path Centrality 53
2.6.1 Centrality 55
2.6.2 Alternative Path Centrality 56
2.7 Multiple Sink Placement Algorithms 56
2.7.1 Minimise the Number of Sinks 58
2.7.2 Fixed Number of Sinks 64
2.7.3 Discussion 69
2.8 Greedy Randomized Adaptive Search Procedures (GRASP) 71
2.9 Summary 73

3 Research Methodology **75**

3.1 Introduction 75
3.2 WSN Model and General Assumptions 75
3.3 WSN Requirements 77
3.4 Simulation Model 79
3.4.1 Input and Output 79

3.4.2 Protocol Stack 80
3.4.3 WirelessPhy Model 81
3.4.4 Radio Propagation Model 81
3.4.5 Simulation Parameters 82
3.5 Performance Metrics 83

4 A Hybrid MAC Protocol for Emergency Response 87

4.1 Introduction 87
4.2 Problem Definition 89
4.2.1 Assumptions 89
4.2.2 Requirements for MAC 90
4.3 ER-MAC Protocol Design 91
4.3.1 Topology Discovery 93
4.3.2 TDMA Slot Assignment 97
4.3.3 Local Time Synchronisation 102
4.3.4 Priority Queue 104
4.3.5 MAC Prioritisation 106
4.3.6 New Nodes 109
4.3.7 Dead Nodes 112
4.3.8 Protocol Overhead 113
4.4 Evaluation of ER-MAC 113
4.4.1 Protocol Comparison 114
4.4.2 Behaviour When a Cluster of Nodes Detects Fire 122
4.4.3 Behaviour Under Variable Traffic Load 125
4.4.4 Behaviour When Topology Changes 127

4.4.5 Behaviour Using Different Topologies 128
4.4.6 Behaviour Using Different Sink Positions 131
4.5 Conclusion 134

5 Fault-Tolerant Relay Deployment for k Vertex-Disjoint Paths 137

5.1 Introduction 137
5.2 Counting-Paths 140
5.2.1 Single Source – Single Sink Problem 141
5.2.2 Multiple Sources – Single Sink Problem 148
5.2.3 Single Source – Multiple Sinks and Multiple Sources – Multiple Sinks Problems 151
5.3 Evaluation of Counting-Paths 151
5.3.1 Single Source – Single Sink Problem 153
5.3.2 Multiple Sources – Single Sink Problem 157
5.4 Greedy Randomised Adaptive Search Procedure for Additional Relay Placement (GRASP-ARP) 161
5.4.1 Construction Phase 163
5.4.2 Node-based Local Search 164
5.4.3 Algorithm Description 164
5.4.4 Acceleration Scheme 167
5.5 Evaluation of GRASP-ARP 167
5.5.1 Multiple Sources – Single Sink Problem 169
5.5.2 Multiple Sources – Multiple Sinks Problem 173
5.6 Conclusion 176

6 Fault-Tolerant Relay Deployment Based on Length-Constrained

Connectivity and Rerouting Centrality **177**

6.1 Introduction 177

6.2 Length-constrained Connectivity and Rerouting Centrality (l-CRC) 179

6.2.1 Length-constrained Connectivity Centrality (l-CC) 180

6.2.2 Length-constrained Rerouting Centrality (l-RC) 182

6.2.3 l-CRC Ranking 183

6.2.4 l-CRC Example 184

6.3 Greedy Randomised Adaptive Search Procedure for Additional Backup Placement (GRASP-ABP) 187

6.3.1 Construction Phase 187

6.3.2 Node-based Local Search 188

6.3.3 Algorithm Description 188

6.4 Evaluation of GRASP-ABP 190

6.4.1 Multiple Sources – Single Sink Problem 192

6.4.2 Multiple Sources – Multiple Sinks Problem 195

6.5 Conclusion 198

7 Multiple Sink and Relay Placement **201**

7.1 Introduction 201

7.2 Multiple Sink Placement (MSP) 203

7.2.1 Greedy Algorithm for Multiple Sink Placement (Greedy-MSP) 204

7.2.2 Greedy Randomised Adaptive Search Procedure for Multiple Sink Placement (GRASP-MSP) 206

7.3 Evaluation of Greedy-MSP and GRASP-MSP 208

7.4 Multiple Sink and Relay Placement (MSRP) 215

7.4.1 Greedy Randomised Adaptive Search Procedure for Multiple Relay Placement (GRASP-MRP) 216
7.4.2 Greedy Algorithm for Multiple Sink and Relay Placement (Greedy-MSRP) 219
7.4.3 Greedy Randomised Adaptive Search Procedure for Multiple Sink and Relay Placement (GRASP-MSRP) 220
7.5 Evaluation of Greedy-MSRP and GRASP-MSRP 224
7.6 Conclusion 234

8 Evaluation of Network Performance 237

8.1 Introduction 237
8.2 Preliminary Discussions and Details of Simulation 238
8.2.1 Preliminary Discussions 239
8.2.2 Details of Simulation 240
8.3 Evaluation of Network Topologies with Multiple Sources and One Sink 241
8.3.1 Experiments Using ER-MAC 242
8.3.2 Experiments Using Z-MAC 247
8.4 Evaluation of Network Topologies with Multiple Sources and Multiple Sinks 249
8.4.1 Evaluation of Network Topologies with Four Sinks 250
8.4.2 Evaluation of Network Topologies with Variable Numbers of Sinks 254
8.5 Conclusion 257

9 Conclusion and Future Work 259

9.1 Summary of Contributions 259

9.1.1 A Hybrid MAC Protocol for Emergency Response 260
9.1.2 Fault-Tolerant Relay Deployment for k Vertex-Disjoint Paths 261
9.1.3 Fault-Tolerant Relay Deployment Based on Length-Constrained Connectivity and Rerouting Centrality 262
9.1.4 Multiple Sink and Relay Placement 263
9.1.5 Evaluation of Network Performance 264
9.2 Future Work 266

Appendices **268**

A Graph Model for WSN **269**
A.1 Notations and Definitions 269
A.2 The Ford-Fulkerson Algorithm 271

B The Disjoint Path Algorithms **273**
B.1 Shortest Vertex-Disjoint Paths with Modified Dijkstra by Bhandari 273
B.2 Fast Pathfinding by Torrieri 274
B.3 Maximum Paths by Torrieri 277

C The Partial k-Connectivity-Repair Algorithm for Relay Placement 279

D The Multiple Sink and Relay Placement Algorithms **283**
D.1 The Multiple Sink Placement Algorithms 283
D.1.1 Minimise the Number of Sinks for Fault-Tolerance (MSFT) . 283
D.1.2 Cluster-Based Sampling for Multiple Sink Placement (CBS-MSP) 285
D.2 The Multiple Sink and Relay Placement Algorithms 287

D.2.1 Minimise the Number of Sinks and Relays for Fault-Tolerance (MSRFT) 287

D.2.2 Cluster-Based Sampling for Multiple Sink and Relay Placement (CBS-MSRP) 289

Bibliography **291**

List of Tables

1 Summary of existing MAC protocols 12

2 Comparison of existing MAC protocols 37

3 Summary of existing relay placement algorithms 42

4 Comparison of existing relay placement algorithms 50

5 Summary of existing multiple sink placement algorithms 59

6 Comparison of existing multiple sink placement algorithms 70

7 Simulation parameters in ns-2 83

8 ER-MAC and Z-MAC simulation parameters in ns-2 115

9 Disjoint paths algorithms' runtime for single source – single sink . . 154

10 Disjoint paths algorithms' runtime for multiple sources – single sink 159

11 Additional relay placement algorithms' runtime for multiple sources – single sink in 25-node networks 171

12 Additional relay placement algorithms' runtime for multiple sources – single sink 172

13 Additional relay placement algorithms' runtime for multiple sources – multiple sinks in 25-node networks 174

14 Additional relay placement algorithms' runtime for multiple sources – multiple sinks 175

15 Additional backup placement algorithms' runtime for multiple sources – single sink 193

16 Additional backup placement algorithms' runtime for multiple sources – multiple sinks 195

17 Multiple sink placement algorithms' runtime with different maximum path length 212

18 Multiple sink placement algorithms' runtime with different sink cost 213

19 Multiple sink placement algorithms' runtime with different average degree 214

20 Multiple sink and relay placement algorithms' runtime with different sink cost 226

List of Figures

1 An overview of a WSN 2

2 WSN protocol stack 8

3 Collision in CSMA 10

4 The hidden terminal problem in CSMA 10

5 Slot structure of S-MAC with fixed duty cycle 14

6 Slot structure of T-MAC with adaptive duty cycle 15

7 The early sleeping problem in T-MAC 15

8 Long preamble in B-MAC 16

9 Latency of B-MAC versus S-MAC 17

10 Power consumption and end-to-end delay of WiseMAC versus CSMA/CA, S-MAC and T-MAC 18

11 Delivery ratio and average delay of TA-MAC versus IEEE 802.11 and S-MAC 20

12 Delivery ratio and delay versus energy efficiency of contention-based MAC protocols and the parameters used 22

13 Delivery ratio and average delay of FLAMA versus TRAMA and S-MAC 25

14 VTS TDMA frame 26

15 Z-MAC frame format 28

16 PMAC frame format 29
17 Funneling-MAC framing 30
18 Crankshaft frame format 32
19 RRMAC superframe structure 33
20 Examples of vertex-disjoint and edge-disjoint paths 38
21 Relay placement problem classification 39
22 Tmote sky 82
23 *TOPOLOGY_DISCOVERY* packet format 94
24 A data gathering tree of six nodes 94
25 Message exchange in topology discovery 95
26 State transition diagram of topology discovery 96
27 Message exchange in TDMA slot assignment 98
28 State transition diagram of TDMA slot assignment 99
29 Schedule packet format 100
30 ER-MAC nodes' transmit schedules 101
31 *SYNCHRONISATION* packet format 103
32 A pair of priority queues 105
33 Data packet format 105
34 ER-MAC's frame structure 107
35 Addition of a new node 111
36 Energy consumption of ER-MAC versus Z-MAC 117
37 Energy consumption of ER-MAC versus Z-MAC for increasing load up to 1 packet/node/sec 117
38 Delivery ratio of ER-MAC versus Z-MAC 118

39 Delivery ratio of ER-MAC versus Z-MAC for increasing load up to 1 packet/node/sec 119
40 Latency of ER-MAC versus Z-MAC 120
41 Latency of ER-MAC versus Z-MAC for increasing load up to 1 packet/node/sec 120
42 Completeness of ER-MAC versus Z-MAC 121
43 A cluster of nodes detects fire 122
44 Energy consumption of ER-MAC versus Z-MAC when a cluster of nodes detects fire 123
45 Delivery ratio of ER-MAC versus Z-MAC when a cluster of nodes detects fire 124
46 Latency of ER-MAC versus Z-MAC when a cluster of nodes detects fire 124
47 Energy consumption of ER-MAC versus Z-MAC under variable traffic load 126
48 Delivery ratio of ER-MAC versus Z-MAC under variable traffic load 126
49 Latency of ER-MAC versus Z-MAC under variable traffic load . . . 127
50 Energy consumption and latency of ER-MAC for network reconnectivity when some nodes die gradually 128
51 Energy consumption and latency of ER-MAC for network reconnectivity when some nodes die simultaneously 129
52 An example of a 100-node network which is easy to partition 130
53 An example of a more robust network with five relay nodes 130
54 Energy consumption of ER-MAC using different topologies 131
55 Delivery ratio of ER-MAC using different topologies 132
56 Latency of ER-MAC using different topologies 132

57 Delivery ratio of ER-MAC using different sink positions 133

58 Latency of ER-MAC using different sink positions 133

59 Counting-Paths example 143

60 Vertex-splitting and external edge replacement in Counting-Paths . 147

61 Number of table lookups versus number of sensor nodes for single source – single sink 153

62 Number of table lookups versus transmission ranges for single source – single sink 155

63 Storage capacity versus number of sensor nodes for single source – single sink 156

64 Number of disjoint paths versus number of sensor nodes for single source – single sink 156

65 Number of disjoint paths versus path length for single source – single sink 157

66 Number of table lookups versus number of sensor nodes for multiple sources – single sink 158

67 Storage capacity versus number of sensor nodes for multiple sources – single sink 160

68 Number of disjoint paths versus number of sensor nodes for multiple sources – single sink 161

69 Number of additional relay nodes needed versus number of disjoint paths required for multiple sources – single corner sink in 25-node networks 170

70 Number of additional relay nodes needed versus number of disjoint paths required for multiple sources – single centre sink in 25-node networks 171

71 Number of additional relay nodes needed versus number of sensor nodes for multiple sources – single corner sink 172

72 Number of additional relay nodes needed versus number of disjoint paths required for multiple sources – multiple sinks in 25-node networks 174

73 Number of additional relay nodes needed versus number of sensor nodes for multiple sources – multiple sinks 175

74 l-CRC example 185

75 Number of backup nodes needed versus number of sensor nodes for multiple sources – single sink 193

76 Number of disjoint paths found for multiple sources – single sink in GRASP-ABP topologies 194

77 Percentage of nodes with disjoint paths for multiple sources – single sink in GRASP-ABP topologies 194

78 Number of backup nodes needed versus number of sensor nodes for multiple sources – multiple sinks 196

79 Number of disjoint paths found for multiple sources – multiple sinks in GRASP-ABP topologies 197

80 Percentage of nodes with disjoint paths for multiple sources – multiple sinks in GRASP-ABP topologies 197

81 Illustration of the MSP problem 203

82 Number of sinks needed for multiple sink placement algorithms versus maximum path length 211

83 Total sink cost for multiple sink placement algorithms versus sink cost 213

84 Number of sinks needed for GRASP-MSP and the optimal solution versus average degree 214

85 Illustration of the MSRP problem 215

86 Total sink and relay cost for multiple sink and relay placement algorithms versus sink cost 226

87 Total numbers of sinks and relays for GRASP-MSRP with *MaxIter* = 1 versus sink cost 227

88 Total numbers of sinks and relays for GRASP-MSRP with *MaxIter* = 10 versus sink cost 228

89 Total numbers of sinks and relays for Greedy-MSRP versus sink cost 228

90 Total numbers of sinks and relays for MSRFT versus sink cost . . . 229

91 Total numbers of sinks and relays for CBS-MSRP versus sink cost . 229

92 Total numbers of sinks and relays for GRASP-MSRP versus maximum path length 230

93 Total numbers of sinks and relays for GRASP-MSRP versus number of candidate relays 231

94 Total numbers of sinks and relays for GRASP-MSRP versus number of candidate sinks 232

95 Total numbers of sinks and relays for GRASP-MSRP versus average degree 232

96 Total numbers of sinks and relays for GRASP-MSRP versus number of nodes 233

97 Delivery ratio of high priority packets for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds 244

98 Delivery ratio of low priority packets for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds 244

99 Latency of high priority packets for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds 245

100 Latency of low priority packets for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds 246

101 Connectivity for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds 247

102 Delivery ratio for multiple sources – single sink with Z-MAC where a node dies every 1,000 seconds 248

103 Connectivity for multiple sources – single sink with Z-MAC where a node dies every 1,000 seconds 249

104 Delivery ratio of high priority packets for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds 251

105 Delivery ratio of low priority packets for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds 251

106 Latency of high priority packets for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds 252

107 Latency of low priority packets for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds 252

108 Connectivity for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds 253

109 Delivery ratio of high priority packets for multiple sources – variable numbers of sinks with ER-MAC where a node dies every 250 seconds 255

110 Latency of high priority packets for multiple sources – variable numbers of sinks with ER-MAC where a node dies every 250 seconds . . 256

111 Connectivity for multiple sources – variable numbers of sinks with ER-MAC where a node dies every 250 seconds 256

Chapter 1

Introduction

Rapid improvements in wireless communication and electronics technologies have enabled the development of small, low-cost, low-power, multifunctional devices, known as sensor nodes. A sensor node (also known as mote) is a battery-powered device with integrated sensing, processing and communication capabilities. A Wireless Sensor Network (WSN) is composed of many sensor nodes, which transmit their data wirelessly over a multi-hop network to data sinks, where data is either processed or transmitted on through a high-speed connection. Unlike sensor nodes which are typically resource-constrained because of a desire to keep them low-cost, small, energy-efficient and easy to deploy, a sink usually has more energy, storage, processing and communication capabilities allowing it to act as a gateway between sensor nodes and an end-user. That is, the sensor readings can simply be relayed by the sink to a database over the Internet. Figure 1 illustrates a typical WSN configuration, which consists of several sensor nodes and a sink. The arrangement and management of a WSN depends on the application for which it is used [10, 46], such as military [53], environmental [74, 105, 15], health [75, 66], home [64] and some commercial applications [30, 29]. These applications require the network to monitor changes in a variety of physical conditions, such as temperature, humidity, light, sound, chemicals, or the presence of certain objects [108] without human intervention.

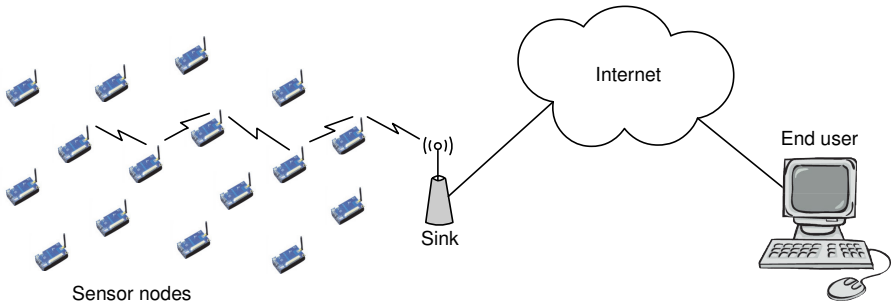


Figure 1: An overview of a WSN consists of several sensor nodes and a sink that acts as a gateway between the sensor nodes and an end-user

Due to the scarce physical resources of sensor nodes, WSN applications must be designed to be energy-efficient, so the operational lifetime can be maximised [12]. Many routing protocols, such as [54, 69, 23], have been designed where energy awareness is an essential design issue. Besides a routing protocol, a Medium Access Control (MAC) protocol also has a major influence on the energy efficiency as it controls the wireless radio, which is the most energy-costly aspect of a sensor node. Both energy-efficient MAC and routing protocols are well-studied in the WSN literature, but their efficiency depends on the physical network topologies that must be well-planned before the actual deployment. In the context of WSN deployment planning, sensor deployment to maximise sensing coverage and guaranteeing connectivity [118, 13, 121], additional relay deployment to improve connectivity [112], and multiple sink placement [85, 8, 122] are also well-studied.

Fault-tolerance is important for many WSN applications as they operate in volatile environments and should remain operational even if some failures occur. WSN failures are for instance caused by dropped packets due to wireless interference, overload, node/link failures, and disconnected networks [18]. To be able to maintain efficient operations, WSNs must be designed to be resilient to these network dynamics. The extreme case would be during emergency response, for example in fire, flood, volcano monitoring and military surveillance. In this case, MAC protocols must be robust to traffic fluctuations and topology changes. Even though reliable routing protocols for WSNs such as those proposed in [24, 33, 47, 125] exist

and are well-understood, the physical network topology must ensure that alternate routes with an acceptable length to the sinks are in fact available when failures occur. This requires a sensor network deployment to be planned with an objective of ensuring some measure of robustness in the topology, so that when failures do occur the protocols can continue to offer reliable delivery.

1.1 Objectives

The primary concern of most WSN communication protocols is energy efficiency, while latency and data delivery rate are typically considered as secondary [123]. However, sometimes the application might need to sacrifice the energy efficiency for a high packet delivery ratio and low latency, for instance when a hazard situation occurs and the WSN needs to monitor it. Since in WSNs these functionalities are controlled by the MAC layer, our first objective is to design a new fault-tolerant MAC protocol that can adapt its behaviour from energy-efficient operation in normal monitoring to reliable and fast delivery in emergency monitoring, and vice versa. This MAC protocol should be able to adapt to traffic and topology dynamics.

Besides requiring a reliable communication protocol, the efficiency and effectiveness of data gathering in an unreliable WSN is also influenced by its physical topology that ensures alternative routes with an acceptable length to the sink are available. Therefore, our second objective is to design fault-tolerant topology planning algorithms, which take into account a path length constraint as sometimes WSN applications have data latency requirements. While in most cases the positions of sensor nodes are predefined because they have to monitor phenomena at certain locations, we aim to find the best locations to deploy additional relay nodes, which do not sense, but only forward data from other sensor nodes. We also aim to deploy multiple sinks for robustness. Because installing many sinks and relays comes at a cost that includes not only the hardware purchase but also the installation and ongoing maintenance, we aim to find the minimal cost deployment.

1.2 Thesis Contributions

1.2.1 Thesis Statement

In this thesis, we demonstrate that medium access control protocols and topology planning algorithms can be designed together to create fault-tolerant wireless sensor networks that trade-off robustness and deployment cost.

1.2.2 Proposed Solutions

The following solutions are the main contributions of this thesis.

1. ER-MAC is a novel hybrid MAC protocol for emergency response WSNs. It tackles the most important emergency response requirements, such as autonomous switching from energy-efficient normal monitoring to emergency monitoring to cope with heavy traffic, robust adaptation to changes in the topology, packet prioritisation and fairness support. Performance evaluation in ns-2 [2] shows the superiority of ER-MAC over Z-MAC [97], a state-of-the-art hybrid MAC protocol, due to its higher delivery ratio and lower latency at low energy consumption.

2. We define a WSN to be robust if at least one acceptable length route to a sink is available for each sensor node after the failure of any $k-1$ nodes. Firstly, we propose the Counting-Paths algorithm to identify the maximum k such that a node has k disjoint paths. Counting-Paths looks for k shortest disjoint paths, where the sum of the lengths plus the spread between the lengths of the k paths is minimal. Secondly, we introduce its dynamic programming variant. Then, we introduce *Greedy Randomised Adaptive Search Procedure for Additional Relay Placement* (GRASP-ARP) that uses Counting-Paths to minimise the number of deployed relays. Empirically, it deploys 35% fewer

relays with reasonable runtime compared to the closest approach from the literature. With the basic Counting-Paths, GRASP-ARP ensures length-bound, while it improves on the runtime with the dynamic programming variant.

3. We look at an alternative solution to reduce the deployment cost by only deploying relays around the most important nodes – those whose failure has the worst effect. To identify such nodes, we define a new centrality measure, *Length-constrained Connectivity and Rerouting Centrality* (l-CRC). We then introduce *Greedy Randomised Adaptive Search Procedure for Additional Backup Placement* (GRASP-ABP) to minimise the number of deployed relays around the most important nodes identified by l-CRC. GRASP-ABP allows us to trade-off the level of fault-tolerance against the runtime.

4. We provide solutions for protecting networks against one sink failure by placing multiple sinks with minimal cost. We look at two heuristics – *Greedy Algorithm for Multiple Sink Placement* (Greedy-MSP) and *Greedy Randomised Adaptive Search Procedure for Multiple Sink Placement* (GRASP-MSP). We show that Greedy-MSP has the shortest runtime, but GRASP-MSP achieves the lowest deployment cost.

5. We propose solutions for protecting networks against either a sensor or a sink failure by deploying multiple sinks and relays with minimal cost. We present *Greedy Algorithm for Multiple Sink and Relay Placement* (Greedy-MSRP) and *Greedy Randomised Adaptive Search Procedure for Multiple Sink and Relay Placement* (GRASP-MSRP). Empirically, GRASP-MSRP's solutions are over 30% less costly than those of Greedy-MSRP and it also has shorter runtime.

6. We evaluate the fault-tolerance of each deployment result by simulations in ns-2 using ER-MAC. In simulation, GRASP-ARP and GRASP-ABP topologies show comparable performance, even though GRASP-ABP deploys fewer relays than GRASP-ARP. For the multiple sink scenario, the best performance is achieved by the GRASP-MSRP topologies due to better sink positions.

1.3 Thesis Overview

The remainder of this thesis is organised as follows.

In **Chapter 2**, we review related work on MAC protocols, algorithms for relay deployment and algorithms for multiple sink deployment.

In **Chapter 3**, we summarise the assumptions we made for the WSN, the simulation tool and the performance metrics.

In **Chapter 4**, we develop ER-MAC, which includes topology discovery, slot assignment, local time synchronisation, priority queue design, MAC prioritisation to cope with large volume of traffic, as well as robust adaptation to new node addition and dead node removal.

In **Chapter 5**, we present Counting-Paths and GRASP-ARP to design topologies with relays. Specifically, we discuss the problems of single source – single sink, multiple sources – single sink, single source – multiple sinks, and multiple sources – multiple sinks.

In **Chapter 6**, we define l-CRC and present GRASP-ABP to design topologies with relays. GRASP-ABP uses l-CRC to identify important nodes in the network.

In **Chapter 7**, firstly we describe the problem of Multiple Sink Placement (MSP). For this problem, we propose Greedy-MSP and GRASP-MSP to design topologies with multiple sinks. Then, we discuss the problem of Multiple Sink and Relay Placement (MSRP), and present Greedy-MSRP and GRASP-MSRP to design topologies with multiple sinks and relays.

In **Chapter 8**, we utilise ER-MAC from Chapter 4 in a data gathering application to evaluate the network performance of topologies that resulted from topology planning algorithms in Chapter 5, 6 and 7.

In **Chapter 9**, we conclude the thesis and identify some further directions for advancing this research.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, we review work related to this thesis. Firstly, we present the general communication architecture of a Wireless Sensor Network (WSN) and review the relevant work on Medium Access Control (MAC) protocols. The comprehensive literature review on MAC protocols contributes toward Chapter 4. Then, we look at algorithms for relay deployment, as well as algorithms for finding disjoint paths, which become the literature study of Chapter 5. We also review the existing centrality and alternative path centrality measurements, which contribute toward Chapter 6. After that, we discuss the relevant literature on multiple sink deployment for Chapter 7 and briefly introduce Greedy Randomised Adaptive Search Procedure (GRASP). The summary of notations and definitions of the graph model for WSNs, which are used in this thesis, can be found in Appendix A.1.

2.2 WSN Communication Architecture

The general WSN communication architecture consists of five layers: the application layer, transport layer, network layer, data link layer, and physical layer, as

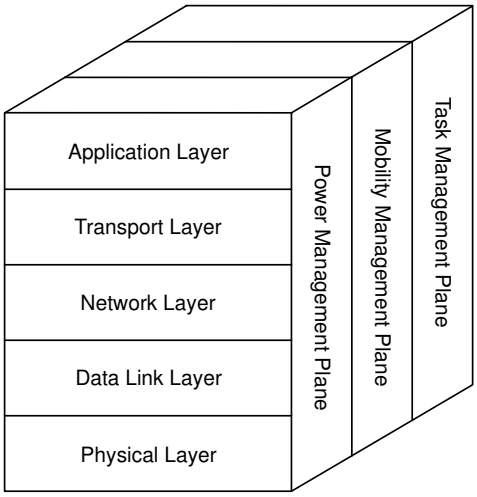


Figure 2: WSN protocol stack [11]

illustrated in Figure 2 [11]. Added to the five layers are the three cross layers planes: power management, mobility management, and task management planes to monitor the power, movement, and task distribution among the sensor nodes, respectively.

1. **Application layer**. In this layer, the application code that is specific for each application is built. It also handles the network management functionalities and query processing to obtain certain information from the network.

2. **Transport layer**. The transport layer protocol helps to maintain the flow of data required by the sensor network application. It provides hop by hop reliability and congestion control.

3. **Network layer**. The main function of this layer is routing the data supplied by the transport layer over multiple hops between nodes that cannot communicate directly.

4. **Data link layer**. MAC protocol resides in this layer to manage channel access policies, scheduling, buffer management and error control.

5. **Physical layer**. The physical layer addresses the modulation, transmission and receiving techniques.

Since WSNs are subject to failures, adaptive and reliable routing has been an active research area. To improve the resiliency of a network, a routing protocol must be able to discover new routes for efficient data delivery to the sinks. However, for a successful communication, not only robust routing protocols, but also reliable MAC protocols are needed. In WSN protocol stack, a MAC protocol resides directly below a routing protocol and one of its fundamental task is to either detect or avoid collision by ensuring that no two nodes within interference range transmit at the same time. It relates to the routing protocol for scheduling of packet transmission purposes in multi-hop networks. A MAC protocol also has a major influence on the energy efficiency of a WSN because it sits directly on top of the physical layer and controls the wireless radio, which dominates the energy consumption of a node, by scheduling its wake-up and sleep periods (duty cycle).

2.3 MAC Protocols for WSN

MAC protocol design may be broadly divided into contention-based and schedule-based medium access [68]. A common contention-based MAC protocol is the Carrier Sense Multiple Access (CSMA) protocol. In CSMA, a node simply senses the channel for a small period of time before transmitting a packet. If it does not sense any activity, it assumes that the channel is clear and starts transmitting the packet. If the channel is busy, the node defers its transmission and continues to monitor the channel until it becomes available.

Figure 3 illustrates a *collision* that is likely to occur in the CSMA protocol. Suppose nodes b, c and d are within node a's transmission range. When node a sends a packet to node b, nodes c and d can overhear a's transmission and wait until it finishes. However, at the end of a's transmission, c and d will transmit their packets to node e simultaneously and cause collision. To prevent collision, CSMA uses *random backoff*, where nodes wait for a random amount of time before they transmit a packet.

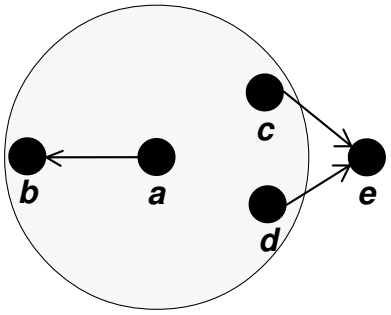


Figure 3: Collision in CSMA: nodes c and d send at the same time after a finishes its transmission

Unfortunately, CSMA protocol also suffers from the *hidden terminal problem*, where collision in any two-hop neighbourhood of a node may occur. Figure 4 illustrates the hidden terminal problem in a network with three nodes. Suppose that nodes a and c can only hear from b, but b can hear from both a and c. When a sends packets to b, c is not aware because it cannot hear a's transmission. If c sends packets when a still transmits, both transmissions will collide at b and all information cannot be retrieved.

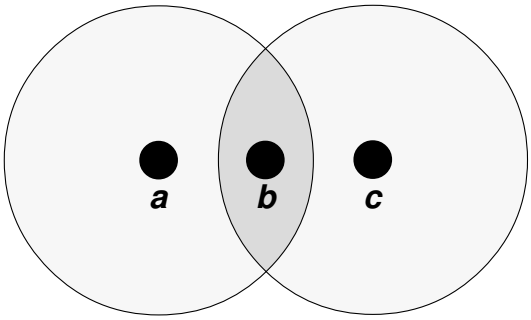


Figure 4: The hidden terminal problem in CSMA: nodes a and c are hidden to each other

CSMA with Collision Avoidance (CSMA/CA) is developed to solve the hidden terminal problem. It exchanges Request-To-Send (RTS) and Clear-To-Send (CTS) packets before data transmission to avoid collision. In Figure 4, if a wants to send data to b, it firstly sends an RTS packet to b. Upon receiving the RTS, b will reply a with a CTS packet. Even if c cannot hear the RTS from a, it can receive the CTS packet from b and refrain from transmitting. The IEEE 802.11 [104] is

a well known contention-based MAC protocol, which is based on the CSMA/CA technique. In the IEEE 802.11, every transmission between a sender and a receiver follows the sequence of RTS/CTS/DATA/ACK, where the receiver must respond with an acknowledgement for each data frame received.

CSMA protocol is popular because of its simplicity and flexibility. It is simple because nodes do not need to synchronise their clocks, and this protocol is able to adapt to traffic fluctuations and changes in node density easily. However, collision in CSMA may still occur when two interfering nodes choose the same random backoff time. In addition, CSMA/CA does not completely eliminate the hidden terminal problem because the RTS packets may also collide. Other sources of energy waste in CSMA are *protocol overhead*, *idle listening* and *overhearing*. Protocol overhead is caused by exchanging RTS and CTS packets before every data transmission. Idle listening occurs when a node turns on its radio and listening to the channel but there are no transmissions. Overhearing happens when a node receives packets that are not intended for it.

Time-Division Multiple Access (TDMA) is a schedule-based MAC protocol that controls the access to the channel by scheduling when a node should transmit, receive, or sleep to conserve energy. TDMA divides time into slots, which are grouped into frames, and requires nodes to be synchronised. The TDMA schedule specifies which slot is to be used by which node to transmit or receive its packet without contending for medium access. The frame size and the slot allocation procedure differ in each TDMA-based protocol, which will be explained in Section 2.3.2. Although TDMA can solve the problems of CSMA, i.e. collision, idle listening and overhearing because of its collision-free schedule guarantee, the scalability issue and the inability to maintain the schedule when the traffic and topology changes are major problems of this protocol.

Many MAC protocols have been designed for WSNs. Below, we present a selection of protocols that have relevance to our problem, i.e. traffic and topology adaptive during emergency monitoring. Based on the mechanisms to access the medium

Table 1: Summary of existing MAC protocols

Protocols	Summary
Contention-based	
S-MAC [123]	Introduces a fixed active-sleep duty cycle.
T-MAC [114]	Modification of S-MAC with adaptive active period.
B-MAC [88]	Uses adaptive preamble sampling to reduce duty cycle.
WiseMAC [39]	Uses a wake-up preamble of minimised size by letting every node learn the sampling schedule of its neighbours.
TA-MAC [48]	Modifies the constant size contention window of S-MAC to be dynamic to current load.
X-MAC [27]	Introduces a series of short preambles with target address to avoid overhearing and reduce energy expenditure on non-target receivers.
MaxMAC [56]	Doubles the duty cycle when the traffic rate reaches the threshold. Switch to CSMA when the traffic rate reaches CSMA threshold.
Schedule-based	
TRAMA [94]	Uses a distributed hash function to determine collision-free slots within two-hop neighbourhood.
FLAMA [93]	Extends TRAMA to reduce idle listening overhead from neighbourhood traffic information exchange.
VTS [38]	Adaptively adjusts the virtual TDMA superframe length based on the number of nodes in range. Reduces the sleep interval if new nodes join the network.
Hybrid	
Z-MAC [97]	Under low contention, nodes can compete in any slots. Under high contention, only the owner of the slot and one-hop neighbours of the owner of the slot can compete for the slot.
PMAC [127]	Adaptively adjusts the sleep/wake-up schedules of the nodes based on its own traffic and the traffic patterns of its neighbours.
Funneling-MAC [7]	Implements CSMA in the entire network with localised TDMA only in the funneling region.
Crankshaft [50]	One frame consists of a number of slots for unicast and for broadcast.
RRMAC [64]	Reduces end-to-end latency by assigning time slots hierarchically and delays ACKs.
EB-MAC [80]	Calculates schedules based on the received signal strength of the detected event.
BurstMAC [98]	Adapts to bursts in traffic by utilising multiple channels for parallel communication.
i-MAC [32]	Assigns the same slot to nodes that have no or low possibility of transmitting together. Assigns different transmission slots to nodes that have high possibility of transmitting together.

for data transmission, we follow the common classification for the MAC protocols: *contention-based*, *schedule-based* and *hybrid* that combines the features of both contention-based and schedule-based protocols. We will analyse the protocols based on their traffic and topology adaptability. Specifically, the key criteria that we address in our reviewed protocols are either delivery rate or throughput, delivery latency of packet transmissions, energy efficiency and fairness over the packets' sources. We seek protocols that are energy-efficient when the traffic load is light, have high delivery rate and low latency when the traffic load increases, and support fairness in both situations. The respective protocols are summarised in Table 1.

2.3.1 Contention-based MAC Protocols

Sensor-MAC (S-MAC)

S-MAC [123] is a slotted protocol that is categorised as contention-based because it contends for the medium before data transmission. S-MAC introduces a fixed duty cycle, which periodically puts nodes into sleep to reduce energy consumption in idle listening. With S-MAC, nodes follow the RTS/CTS/DATA/ACK sequence to avoid collision. When some nodes want to send data, they independently contend for the medium using RTS packets. The first sender whose RTS packet reaches the intended receiver wins the medium and the receiver replies the sender with a CTS packet. After starting data transmission, nodes do not follow their sleep schedule until they finish the transmission. S-MAC avoids overhearing, which is one of the major sources of energy waste, by letting neighbours of the sender and the receiver go to sleep after hearing RTS/CTS packets.

S-MAC requires nodes to be synchronised, so they have the same listen/sleep schedule. These schedules coordinate nodes to minimise additional delivery latency. A node first listens for a certain amount of time. If it does not hear a SYNC from another node, it broadcasts a SYNC packet that includes its address and the time of its next sleep. Receivers of the SYNC will adjust their timers immediately after

receiving the SYNC packet. This method helps new nodes join the network later by listening to the SYNC packet that is broadcast regularly to inform the common schedule. The slot structure of S-MAC with the fixed duty cycle is shown in Figure 5.

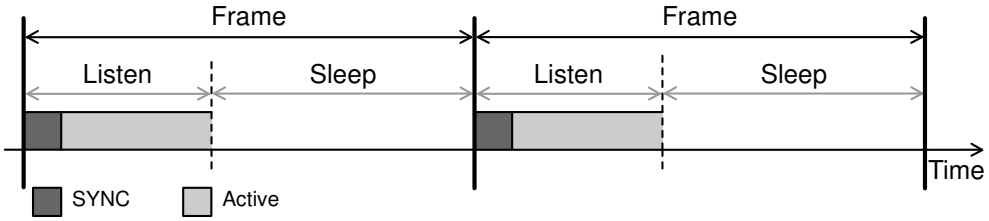


Figure 5: Slot structure of S-MAC with fixed duty cycle

S-MAC trades off latency for energy efficiency by periodically putting nodes to sleep. Therefore, it does not adapt to the changes of the traffic very well. Under a heavy traffic, the latency increases due to the periodic sleep of each node, which is accumulated at each hop. S-MAC also sacrifices fairness by letting a node who has more data get more time to access the channel. The simulations in [123] only measure the energy consumption of source and intermediate nodes. The results are compared to 802.11. For the source nodes, 802.11 uses twice as much energy as S-MAC under heavy traffic and three times under light load. For the intermediate nodes, S-MAC consumes around 15% more energy than 802.11 during heavy traffic, but consumes around 40% less energy when the traffic is light.

Timeout-MAC (T-MAC)

T-MAC [114] is a slotted protocol that tries to improve on S-MAC by using an adaptive duty cycle, which dynamically adjust nodes' sleep and active cycles based on communication of neighbouring nodes. T-MAC ends the active time using time out on hearing nothing. Simulations in [114] have shown that under variable loads, T-MAC's energy consumption outperforms S-MAC by a factor of 5. Similarly to S-MAC, nodes in T-MAC communicate using RTS/CTS/DATA/ACK. The slot structure of T-MAC is shown in Figure 6.

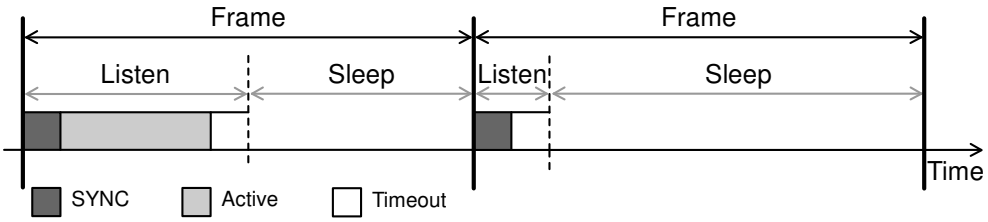


Figure 6: Slot structure of T-MAC with adaptive duty cycle

Unfortunately, it is common to all contention-based protocols including T-MAC that the chance of collision increases rapidly during high traffic loads. Moreover, T-MAC's early sleeping problem effectively increases latency and reduces throughput. This problem is illustrated in Figure 7. Suppose messages flow from node a to node d. When node c overhears a CTS packet from node b to node a, it remains silent. Node d is not aware of the communication between a and b, and hear nothing from c, so the timeout forces d to sleep early. When c wants to send data to d, it has to wait until the next wake-up schedule.

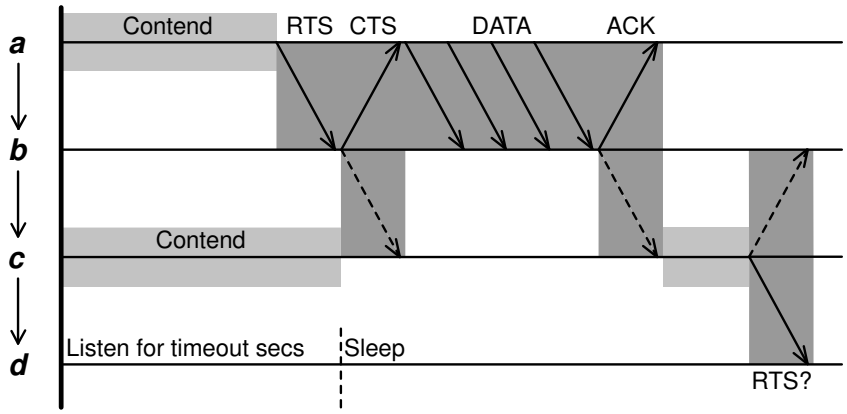


Figure 7: The early sleeping problem in T-MAC: d goes to sleep before c can send an RTS to it [114]

Berkeley-MAC (B-MAC)

B-MAC [88] is an asynchronous duty-cycled protocol that introduces adaptive preamble sampling to minimise the active duration of receivers, and thus reduces duty cycle and minimises idle listening. As an asynchronous duty-cycled protocol,

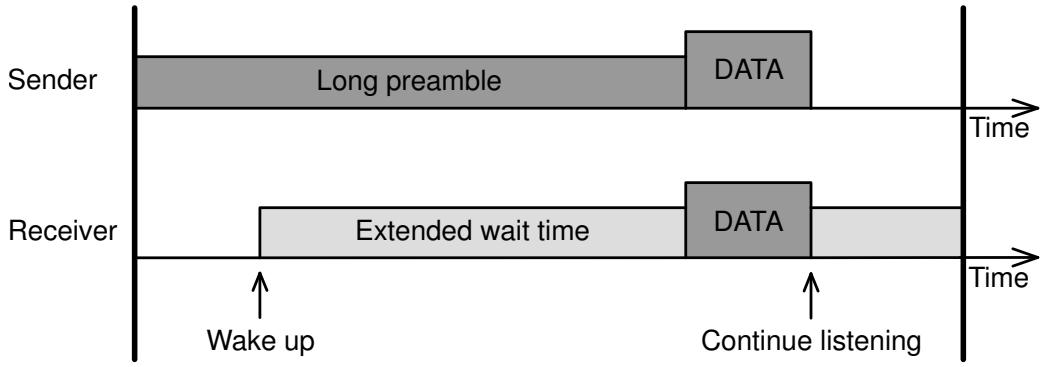


Figure 8: Long preamble in B-MAC

B-MAC does not wake up nodes simultaneously. Therefore, before sending data to a receiver, a sender sends a long preamble, which has the same length as the channel check interval, i.e. the interval where a node periodically wakes up and checks for activity on the channel. Using a long preamble allows the receiver to wake up, detect activity on the channel, receive the preamble and then receive the data.

B-MAC is the first Low Power Listening (LPL) protocol. In this scheme, each time a receiver wakes up, it turns on the radio and checks for any activity on the channel. If an activity is detected, the receiver powers up and stays awake to receive incoming packets as shown in Figure 8. Otherwise, a timeout forces it back to sleep. Idle listening still occurs in B-MAC when the node wakes up but there is no activity on the channel. B-MAC is able to assess whether the channel is clear or busy, which is referred to as Clear Channel Assessment (CCA). It searches for the outliers in the received signal such that the channel energy is significantly below the noise floor. If an outlier exists, the channel is clear. If five samples are taken and no outlier is found, the channel is busy. B-MAC takes five samples of signal strength to reduce the possibility of false alarms because of random noise. This protocol also uses random backoff when sending a packet.

The simulation results in [88] compare the performances of B-MAC to S-MAC. Without duty cycling, B-MAC achieves over 4.5 times the throughput of S-MAC

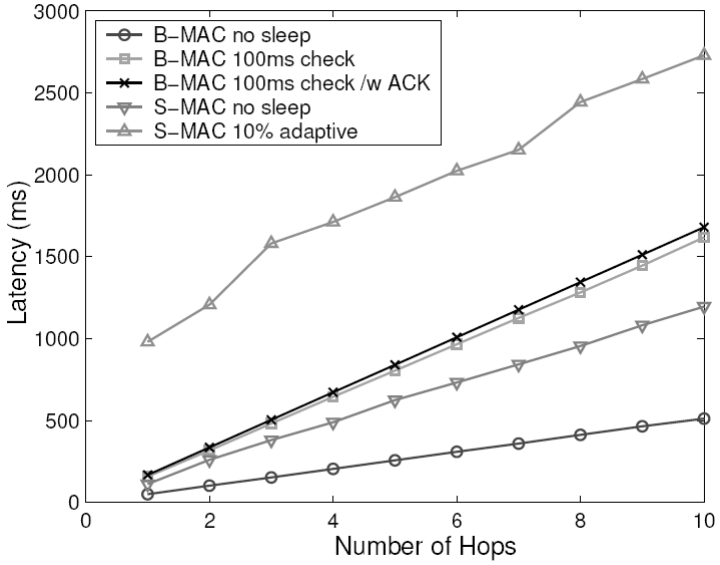


Figure 9: Latency of B-MAC versus S-MAC [88]

because S-MAC has overhead from RTS/CTS exchanges. The energy consumption of B-MAC under heavy loads is 50% less than S-MAC. However, B-MAC consumes 25% more energy than S-MAC when the traffic is light because the long preamble dominates the energy usage. To test the latency, 11 nodes are arranged to form a 10-hop linear topology, where the source node and the sink node are separated 10 hops away. The end-to-end latency of B-MAC versus S-MAC is shown in Figure 9. As a contention-based protocol, B-MAC provides flexibility to handle dynamic changes in node densities. However, the use of a long preamble before every data transmission contributes to additional latency at each hop, especially when the traffic load increases.

Wireless Sensor MAC (WiseMAC)

WiseMAC [39] is a MAC protocol based on the preamble sampling technique. In this technique, each node listens to the channel for a short duration for possible reception of packets. If the medium is busy, the receiver continues to listen until a data frame is received or until the medium becomes idle again. If a node wants to send data, it adds a wake-up preamble in front of every data frame to ensure that

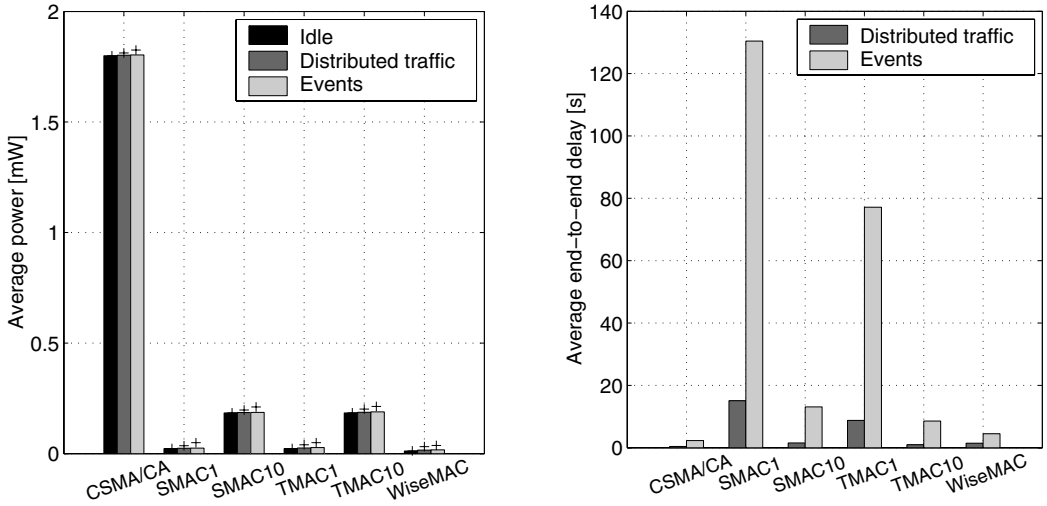


Figure 10: Power consumption and end-to-end delay of WiseMAC versus CSMA/CA, S-MAC and T-MAC [39]

the receiver will be awake when the data portion of the packet arrives. The novelty of WiseMAC is using a wake-up preamble of minimised size by letting every node learn the sampling schedule of its neighbours. The first communication between two nodes always uses a long wake-up preamble of size equal to the sampling period. However, when the schedule is acquired, a wake-up preamble of reduced size can be used. WiseMAC is traffic adaptive because the length of the wake-up preamble is proportional to the packet inter-arrival time, which is small when the traffic is high.

WiseMAC does not need the whole network to be synchronised. An acknowledgement packet in WiseMAC is not only used to acknowledge the reception of a data packet, but also to inform other nodes of the remaining time until the next sampling time. In this way, a node can keep a table of sampling time offsets of all its usual destinations up-to-date. WiseMAC also has a *more bit* in the header of data packets. When a data packet is received with the more bit set, the receiver will continue to listen for another data packet after having sent the acknowledgement. The sender will send the following data packet right after having received the acknowledgement.

Figure 10 show the power consumption and the end-to-end delay of WiseMAC, CSMA/CA, S-MAC and T-MAC during *idle*, *distributed traffic* and *event* experiments. There is no traffic in the idle experiment. In the distributed traffic and event experiments, the source nodes generate traffic not at the same time and at the same time, respectively. The CSMA/CA protocol provides the lowest average latency with the highest power consumption compared to all other protocols. S-MAC and T-MAC with 1% duty cycle provide a low average power consumption, in the order of what is provided by WiseMAC. However, the corresponding latency is very high, while it remains low for WiseMAC. S-MAC and T-MAC with 10% duty cycle are able to provide a relatively low latency, but at the price of a power consumption that is much higher than the one of WiseMAC.

The disadvantages of this protocol are that the long wake-up preambles cause throughput limitation and large power consumption overhead in transmission and reception. Moreover, the decentralised listen/sleep schedules result in different sleep and wake-up times for each neighbour of a node. This contributes to additional per-hop latency and causes the hidden terminal problem, where a preamble from a node can interfere with a data packet from another node that is not within range.

Traffic Adaptive-MAC (TA-MAC)

Another contention-based protocol that tries to improve on S-MAC is TA-MAC [48]. It modifies the constant size contention window of S-MAC to reduce the probability of collision. TA-MAC introduces dynamic contention window according to the current traffic load of the network. The current load is defined as the average number of collision. A sender detects a collision if it does not receive an acknowledgement after sending a data packet.The average number of collision is calculated as a combined weight metric of the previous average number of collision and the current number of collision. If the average number of collision exceeds a threshold, TA-MAC assumes the traffic load has increased and doubles the contention window. Otherwise, it knows that the traffic has decreased and the contention window is halved.

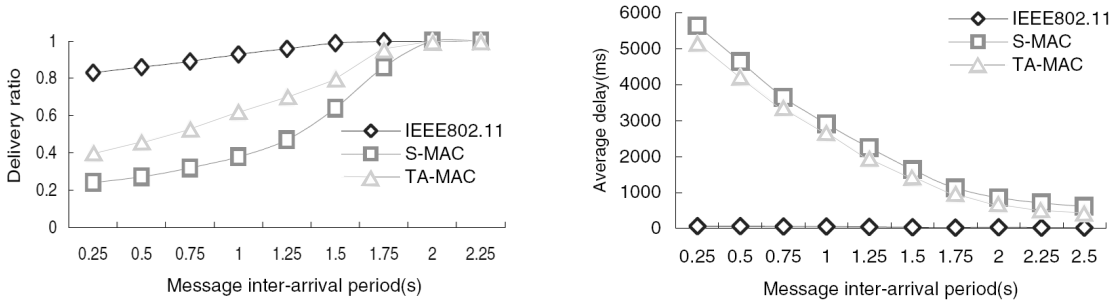


Figure 11: Delivery ratio and average delay of TA-MAC versus IEEE 802.11 and S-MAC with 10% duty cycle [48]

TA-MAC achieves energy savings by reducing the time spent in idle listening during backoff procedure after collisions happen. The simulation results in [48] have shown that TA-MAC achieves 15%–20% energy savings compared to S-MAC with 10% duty cycle. It also has a slightly lower latency and higher delivery ratio than S-MAC when the traffic load is high as depicted in Figure 11. This is due to the fact that TA-MAC's probability of collision is lower than that of S-MAC. Even though TA-MAC improve on S-MAC, it inherits S-MAC's limitation on the additional per-hop latency due to periodic sleep at each node, which is not acceptable for emergency monitoring. In addition, collisions still occur during heavy traffic after the contention window reaches its maximum value and cannot be extended any further.

X-MAC

X-MAC [27] is an asynchronous duty-cycled MAC protocol that tries to solve the problems of standard asynchronous duty-cycled MAC, such as B-MAC. The standard asynchronous duty-cycled MAC has a long preamble that increases latency and energy consumption at both senders and receivers. X-MAC introduces a series of short preambles with target address to avoid overhearing of low power listening and to reduce the energy expenditure on non-target receivers. X-MAC also inserts pauses between short preambles. This enables the target receiver to send an early acknowledgement and thus shorten the preamble period. This scheme is claimed

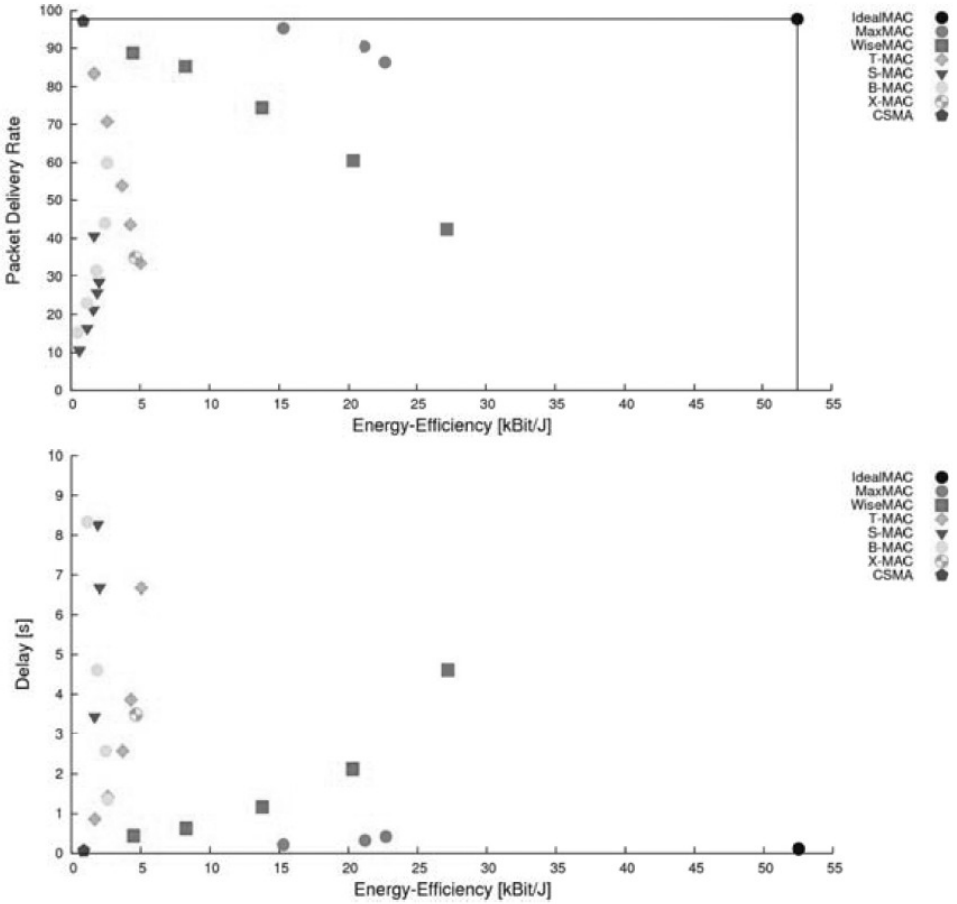
to achieve further energy savings and reduce latency. However, waiting for an acknowledgement from a receiver before sending a data packet has a disadvantage. A sender will fail to send its data if its preambles are not acknowledged by the intended receiver within a given time interval. X-MAC also utilises random back-off when a sender node wants to send data but it detects a preamble. However, this does not solve the hidden terminal problem, because collision still occurs when several backed off senders send their data without preambles.

X-MAC performances are compared to a basic asynchronous Low Power Listening (LPL) MAC protocol. Unlike the LPL protocol, X-MAC's energy consumption is relatively constant with increasing network density. The LPL protocol consumes more energy when the network density increases. X-MAC also has 50% lower latency and higher delivery ratio compared to the LPL protocol because LPL has longer preamble.

MaxMAC

Hurni and Braun [56] introduce MaxMAC, a traffic adaptive MAC protocol that targets at higher throughput, lower latency and lower energy consumption. MaxMAC uses WiseMAC's preamble sampling scheme and X-MAC's overhearing avoidance by adding target address in the preamble. When the rate of incoming packets reaches a predefined threshold T_1, a receiver doubles the duty cycle and stays in the new state S_1 for a predefined *LEASE* timespan. The receiver informs the sender about this change through an acknowledgement. When the rate of incoming packets reaches a threshold T_2, the receiver enters the state S_2 and doubles the duty cycle again. Then, when the rate of traffic reaches a further threshold T_{CSMA} ($T_{\text{CSMA}} > T_2 > T_1$), MaxMAC switches to CSMA state.

MaxMAC does not introduce any control messages. All necessary control information is communicated in the header of data frames as well as in acknowledgment frames. When the traffic is light, MaxMAC is energy-efficient as other energy-efficient MAC protocols, but it falls to CSMA energy consumption to achieve



MaxMAC		B-MAC	
Base Interval	100, 200, 250 ms	Base Interval	25, 50, 100,
Duty Cycle	2, 1, 0.8%		200, 500 ms
LEASE	1 s	Duty Cycle	8, 4, 2, 1, 0.4%
T_1, T_2, T_{CSMA}	4, 8, 12 packets/s	WiseMAC	
S-MAC		Base Interval	25, 50, 100,
Listen Interval	100, 200, 300, 500		200, 500 ms
	1000, 2000 ms	Duty Cycle	8, 4, 2, 1, 0.4%
Duty Cycle	10%	Medium Reservation	u[0,10] $\times$ t_{rx-tx}
T-MAC		X-MAC	
Frame Length	50, 100, 200 ms	Max Interval	200 ms
	300, 500 ms	Min Interval	10 ms
SYNC & RTS size	14 bytes	EarlyACK size	10 bytes
CTS size	10 bytes	CSMA	
SYNC period	10 s	Contention Window	10 ms

Figure 12: Delivery ratio and delay versus energy efficiency of contention-based MAC protocols and the parameters used [56]

higher throughput and lower latency when the traffic bursts. As a combination of WiseMAC and X-MAC, it inherits their limitations, where the use of wake-up preambles contributes to additional per-hop latency and may interfere with data transmissions (hidden terminal problem).

Figure 12 show the comparisons of delivery ratio and delay of the reviewed contention-based MAC protocols and the simulation parameters used in [56]. The IdealMAC protocol is used as a reference to show the lower bounds of network performance. With IdealMAC, nodes always know when they need to switch to receive/transmit in order to handle data transmissions. IdealMAC has the lowest-possible latency, the highest possible energy efficiency, does not suffer from overhearing or idle listening, and can always avoid collisions. In this simulation, one event is triggered every 30 seconds at a random location of the simulated 49-node network and only nodes that can sense the event are required to generate traffic. The simulation results show that MaxMAC outperforms the other contention-based protocols in this scenario.

2.3.2 Schedule-based MAC Protocols

TRaffic Adaptive Medium Access (TRAMA)

TRAMA [94] is designed for periodic data collection and monitoring applications. It organises time into random access and scheduled access slots. Nodes send signaling packets for neighbour discovery and time synchronisation during the random access slots and data packets during the scheduled access slots. TRAMA uses a distributed hash function to schedule collision-free slots for data transmissions. Every data packet contains a summary of a node's schedule. Therefore, each node has to listen to the last data messages from its one-hop neighbours in order to synchronise its schedule with theirs. With TRAMA, nodes are allowed to exchange information about network topology and traffic conditions regularly in their two-hop neighbourhood. Based on this information, TRAMA uses a distributed election scheme to

determine the state of a node, i.e. transmit, receive, or sleep. Although TRAMA adapts to topology changes by utilising CSMA periods to allow new nodes to join the network, it only provides limited capabilities to adjust to traffic fluctuations. This is achieved by allowing a node to release its slot to be used by other nodes if it has no data to send.

Compared to S-MAC with 10% duty cycle, TRAMA is more energy-efficient. S-MAC has a fixed duty cycle, so it has a constant percentage of sleep time, i.e. around 80% during both light and heavy traffic. Nodes with TRAMA sleep 6% more than S-MAC when the traffic is light, but wake up 18% more when the traffic load is high to achieve high delivery ratio. The simulation results in Figure 13 have shown that TRAMA achieves higher delivery ratio, which is around 40% to 60% over S-MAC when the traffic load increases. However, as other schedule-based protocols, TRAMA suffers higher latency, which is around 10 times higher than S-MAC.

FLow-Aware Medium Access (FLAMA)

FLAMA [93] is a schedule-based MAC protocol that extends TRAMA in an attempt to reduce idle listening overhead from neighbourhood traffic information exchange. TRAMA requires nodes to exchange traffic information regularly to maintain schedules during the scheduled access periods. Unlike TRAMA, FLAMA exchanges traffic information implicitly only during the random access periods. FLAMA uses a data gathering tree, where a node has incoming traffic flows from all its children and it has only one outgoing flow to its parent. Because of this predictable traffic pattern, FLAMA uses *flows* to represent one-hop traffic information and to specify the senders, the receivers and the packet's rate. The traffic information of a node is determined based on a function of incoming flow rates from its children. These flows are used to set up transmit, receive and sleep schedules of nodes in the network using a distributed election algorithm. In FLAMA, nodes that produce or forward more traffic are assigned more slots.

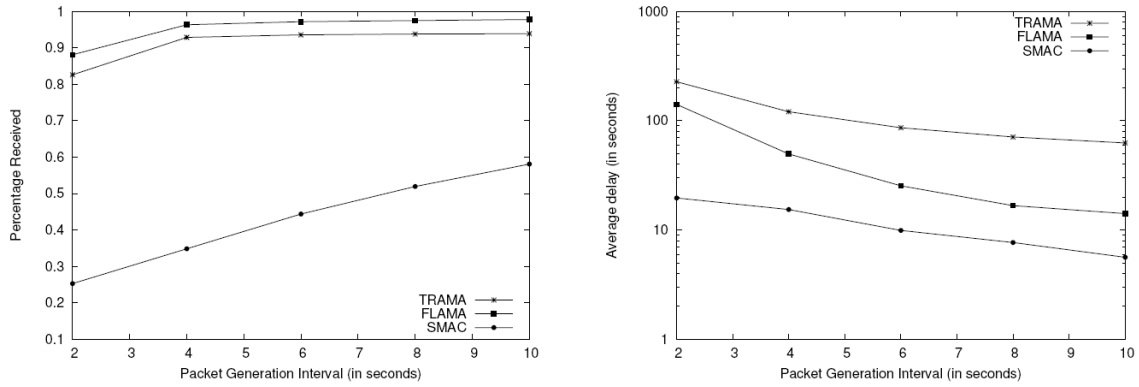


Figure 13: Delivery ratio and average delay of FLAMA versus TRAMA and S-MAC with 10% duty cycle [93]

Simulations conducted in [93] have shown that FLAMA outperforms TRAMA and S-MAC with 10% duty cycle for better delivery ratio and energy consumption. Nodes with FLAMA sleep around 85% regardless of the traffic loads, but achieve 5% higher delivery ratio than TRAMA. However, S-MAC outperforms these two protocols in terms of average latency as depicted in Figure 13.

Virtual TDMA for Sensors (VTS) MAC Protocol

VTS [38] is designed for soft real-time applications, where a packet has a bounded latency. This protocol adaptively adjusts a *virtual* TDMA superframe (set of frames) length according to the number of nodes in range. Virtual means that nodes know neither superframe limits nor their relative position in the superframe. They only know that they can transmit packets every superframe length cycle. Using the flexible superframe length, VTS allows nodes to join or leave the network easily. When new nodes join the superframe, latency increases. Thus, VTS tries to keep the latency below a given threshold value by reducing the sleep interval (which corresponds to increasing the duty cycle).

This protocol assumes a network with a single-hop cluster. Therefore, there is a sink to start the setup by broadcasting a control (CTL) packet with an initial predefined value of the duty cycle. The sink then dynamically adjusts the superframe length

by recomputing the new duty cycle value based on the number of nodes belonging to the superframe and informs it to the nodes with its CTL packet. VTS uses the CSMA/CA mechanism for data delivery, where a unicast transmission follows the RTS/CTS/DATA/ACK sequence. The TDMA frame of VTS is illustrated in Figure 14. A CTL packet can be used as a SYNC, an RTS or a keep-alive beacon.

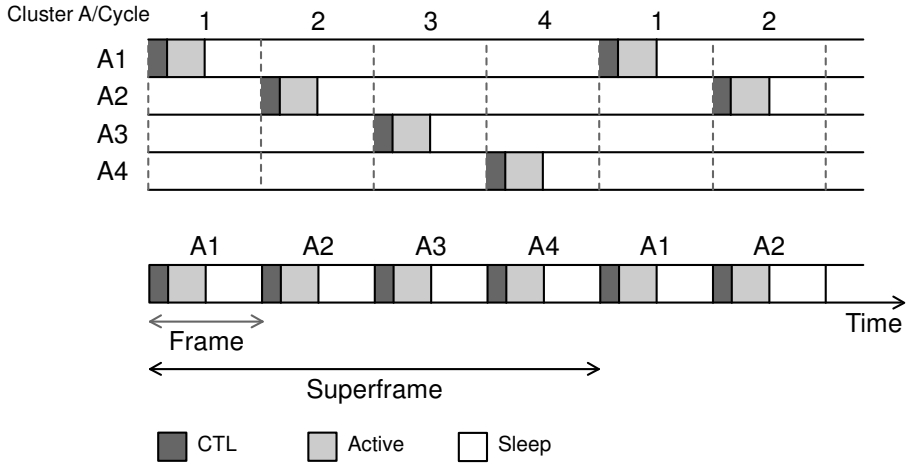


Figure 14: VTS TDMA frame

VTS sacrifices average latency for energy efficiency at low loads, but guarantees latency at high loads. VTS is compared to 10% duty-cycled S-MAC with and without adaptive listening. With adaptive listening, nodes which overhear an RTS or CTS packet wake up at the end of the transmission, instead of waiting for their next schedules. This scheme achieves higher throughput and lower average latency at the cost of higher energy consumption. The results in [38] have shown that VTS has the lowest maximum latency compared to the two types of S-MAC at high loads, but S-MAC with adaptive listening has the lowest average latency. S-MAC with and without adaptive listening suffer 8 and 15 times the maximum latency of VTS, respectively, because the latency of VTS never exceeds the superframe length. VTS's throughput is slightly better than S-MAC without adaptive listening. The throughput of S-MAC with adaptive listening is three times better than VTS at high loads, but its power consumption is twice as much as VTS. In addition, VTS consumes less energy than the two types of S-MAC at both high and low loads.

2.3.3 Hybrid MAC Protocols

Hybrid MAC protocols are protocols that combine the features of both contention-based and schedule-based mechanisms. It takes advantages of the simplicity and flexibility of the contention-based scheme and the collision-free nature of the schedule-based one. We review the hybrid MAC protocols that are designed to be traffic adaptive and have the ability to cope with topology changes too.

Zebra-MAC (Z-MAC)

Z-MAC [97] is a hybrid MAC protocol that dynamically switches between CSMA and TDMA depending on the traffic load in the network. Each node gets a collision-free TDMA slot by executing a distributed slot selection algorithm based on its two-hop neighbourhood schedule information. Z-MAC's frame format is depicted in Figure 15. Each slot begins with a small contention period where nodes can compete to access the channel. When competing for the channel access, the slot owner has a higher priority to transmit than the non-owners if it has data to send. If the slot owner has no data to send, it allows other nodes to use its slot. For example, if a node has data to send and it is the owner of the slot, it back offs within T_{o} period, else it back offs between T_{o} and T_{no}. Then, it performs CSMA (using B-MAC's random access approach) and if the channel is clear, it sends its data.

Under Low Contention Level (LCL), nodes in the network can compete in any time slots and thus achieve high channel utilisation and low latency. However, under High Contention Level (HCL), only the owner of the slot and one-hop neighbours of the owner of the slot can compete for the slot. Therefore, it reduces collisions. Z-MAC determines the state of low or high contention based on packet loses due to hidden terminals. In high contention networks, Z-MAC uses Explicit Congestion Notification (ECN) messages to reduce hidden terminals. When a sender node detects heavy traffic loads, it broadcasts the ECN message to its one-hop neighbours

and they propagate the ECN message to their neighbours. Therefore, nodes within two-hop neighbourhood of the sender are notified of the current traffic. These neighbours can only compete for its scheduled slot and its direct neighbours' slots, but return to their previous states when they receive no more ECN messages.

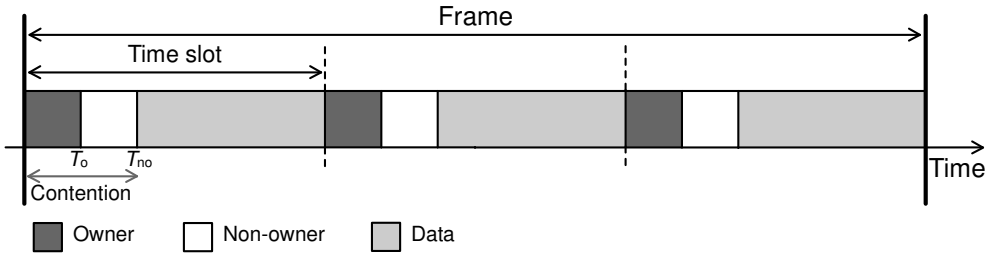


Figure 15: Z-MAC frame format

Switching states from LCL to HCL and vice versa make Z-MAC adaptive to traffic fluctuations. It also adapts to topology changes because the distributed slot selection algorithm can performs a localised time slot assignment for new nodes. Z-MAC builds a TDMA structure on top of CSMA, using B-MAC's backoff mechanism, Clear Channel Assessment (CCA) and Low Power Listening (LPL) to reduce energy consumption. Hence, it inherits B-MAC's limitation in idle listening. Results in [97] have shown that Z-MAC achieves 20%–30% higher throughput compared to B-MAC under high contention, even though B-MAC's throughput is slightly better (5%–10%) than Z-MAC under low contention. In terms of energy efficiency, Z-MAC is slightly worse (10%) than B-MAC under low traffic rate, because of the larger backoff window size and the use of synchronisation messages. However, its energy efficiency improves under high traffic rate and beats B-MAC by 40%.

Pattern-MAC (PMAC)

PMAC [127] combines CSMA and TDMA, and adaptively adjusts the sleep/wake-up schedules of the nodes based on its own traffic and the traffic patterns of its neighbours. Based on expected traffic patterns, a node can sleep for several time frames when there is no traffic in the network. If there is any activity in the local

neighbourhood, the node will know this through the patterns and will wake up when required. A pattern is basically a repetition of n times sleep and one wake-up cycle. Figure 16 illustrates PMAC's frame format. A frame consists of pattern repeat period for data transmission and pattern exchange period for pattern exchange task. The numbers of slots in the pattern repeat period and the pattern exchange period depend on the user's application and the maximum number of neighbours a node could have, respectively. During the pattern repeat period, a node follows its sleep/wake-up schedule and transmits data using RTS/CTS/DATA/ACK mechanism. Then, during the pattern exchange period, every node contends for the channel access to announce its pattern information. If during this period, a node does not receive new pattern information, it will use the old traffic pattern for the next frame.

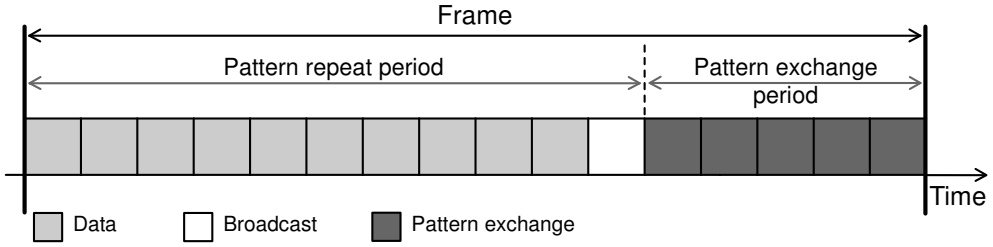


Figure 16: PMAC frame format

The main advantage of PMAC is the adaptability to traffic changes. The simulations conducted in [127] have shown that PMAC can achieve the same throughput as 10% duty-cycled S-MAC at light loads and seven times higher at high loads, with 30% less total energy consumption at any traffic loads. However, using traffic pattern leads PMAC to be more prone to error. For example, some nodes may receive incorrect patterns due to interference signals while other nodes successfully receive the correct pattern. This problem increases energy consumption because of collision, idle listening and wasted transmissions. Moreover, the fixed numbers of slots for both pattern repeat and pattern exchange periods make PMAC less robust to topology changes, especially when the network requires addition of new nodes beyond the numbers of pre-assigned slots.

Funneling-MAC

Funneling-MAC [7] tries to solve the problem of increasing packet loss at nodes closer to the sink, which is known as the funneling region, even at low traffic loads. It implements CSMA in the entire network with a localised TDMA algorithm overlaid only in the funneling region. Funneling-MAC localised TDMA is triggered by a beacon broadcast by the sink. Any nodes receiving this beacon become *f-nodes* and synchronise with other f-nodes by initialising their clock. Funneling-MAC is sink-oriented, because the sink monitors the traffic, calculates and broadcasts the TDMA schedule using the same transmission power as sending beacons. The sink computes the TDMA schedule by allocating time slots per path basis and taking into account slot reuse within more than two hops. The number of slots given to a path is equivalent to the traffic rate of the path and the number of hops in the path. To enhance robustness and flexibility, a CSMA frame is reserved between two TDMA frames, and carrier sense is performed even for scheduled transmission. In funneling-MAC, the superframe duration is fixed while the TDMA duration changes dynamically according to the current traffic rate. Figure 17 illustrates the funneling-MAC framing.

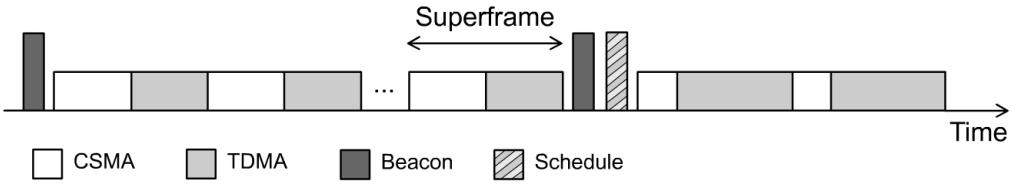


Figure 17: Funneling-MAC framing

Funneling-MAC is traffic adaptive because the sink can increase its transmission power to send beacons and schedules, which in turn increases the size of the funneling region, when the traffic load increases. It also topology adaptive because of the CSMA mechanism. Funneling-MAC is built on top of B-MAC and uses B-MAC's Low Power Listening (LPL) and preamble technique. Nodes outside the funneling

region, which communicate using CSMA, use a long preamble before data transmission, while nodes in the funneling region with TDMA use a short preamble. This makes funneling-MAC inherit B-MAC's limitation in idle listening, especially for nodes outside the funneling region. Moreover, concentrating on the traffic inside the funneling region makes it difficult for funneling-MAC to control high data contention at nodes outside the intensity region.

The simulation results in [7] have shown that compared to B-MAC, funneling-MAC achieves lower loss rate at nodes around one and two hops from the sink. At high traffic rate, B-MAC's loss rate is 81% at one hop and 40% at two hops from the sink, while funneling-MAC only loses 48% at one hop and 22% at two hops. Funneling-MAC achieves higher throughput than B-MAC and Z-MAC at low loads (around 50%) and medium loads (around 30%), but shows similar performance at high loads.

Crankshaft

Another protocol that combines CSMA and TDMA is Crankshaft [50]. This protocol is designed for dense WSNs. Similar to PMAC, Crankshaft schedules receive slots. It tries to reduces overhearing by offsetting wake-up schedules. Crankshaft divides time into frames and each frame consists of a number of slots for unicast and broadcast communication. The frame format of Crankshaft is illustrated in Figure 18. This protocol allocates one unicast slot in a frame for every node in the network based on the nodes MAC address. By doing so, a sender knows precisely when the intended receiver wakes up. Any senders that want to send message to a node contend for the channel. Since a node's receive slot is calculated as the node's MAC address modulo the number of unicast slots in a frame, two neighbours may be assigned the same slot. To allow the two neighbours to communicate, nodes are allowed to act as senders in their own receive slot, but revert to receive mode if they lose contention. During the unicast period, each node listens in one slot but sinks, which are assumed to be more powerful than sensor nodes, listen to all unicast slots. Then, during the broadcast period, all nodes wake up to listen.

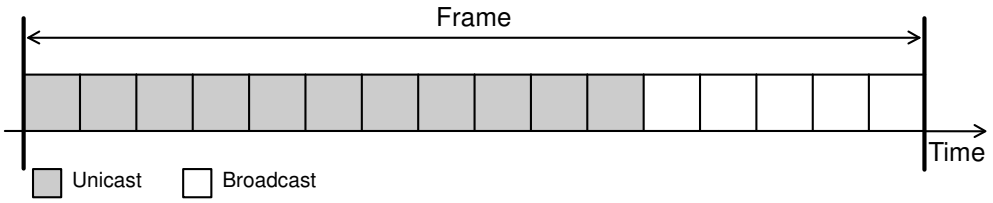


Figure 18: Crankshaft frame format

Crankshaft is suitable for long-lived monitoring application, where throughput can be traded for energy efficiency. Crankshaft has 10% lower delivery ratio but 20% lower latency and consumes a factor of 8 less energy than a basic Low Power Listening (LPL) MAC protocol.

Real time and Reliable MAC (RRMAC)

RRMAC [64] is a hybrid MAC designed for hard real-time and reliable communication, where a successful communication depends on the performance time, otherwise the system fails. This protocol uses convergecast network with multiple single-hop clusters and assumes that sensor nodes have shorter transmission range than base station and sinks (cluster heads). Therefore, the base station can communicate directly to sinks and sink can communicate directly to sensor nodes within one cluster. RRMAC attempts to reduce end-to-end latency in two ways. First, time slots are assigned in a sequence so that a packet can flow continuously from leaf node to the base station and the base station can get all data in one superframe duration. Second, RRMAC delays acknowledgment. Receiver may send acknowledgment upon successful transmission in the next beacon frame.

The time slot assignment in RRMAC is hierarchical, where the base station allocates time slots to sinks and sink assigns time slots to sensor nodes. This mechanism makes RRMAC topology adaptive because sensor nodes' TDMA schedules for single-hop communication are assigned periodically. Sensor nodes and sinks transmit latency sensitive data to their parents using the assigned time slots. If

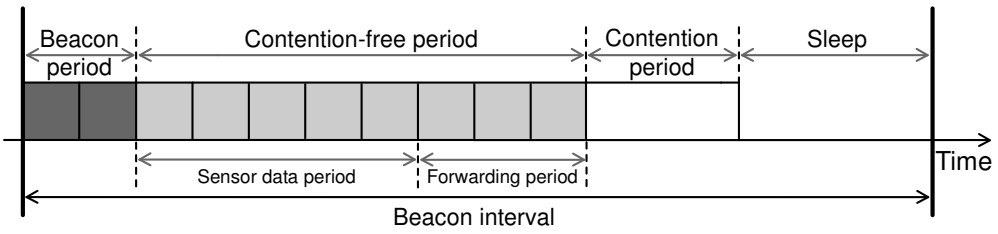


Figure 19: RRMAC superframe structure

sensor nodes or sinks do not need real time or reliable data transmission, the superframe structure will contain only a beacon block and contention access period. With RRMAC, all sensor nodes are synchronised during beacon period, which is owned by the base station and sinks. RRMAC's superframe structure is illustrated in Figure 19.

The simulation results in [64] have shown that when the traffic is stable, RRMAC achieves constant delivery ratio and latency, which is beneficial for real-time requirements. When the traffic load increases, RRMAC can utilise the contention access period for data transmission. However, having only one contention period in every superframe while the TDMA slots are contention-free makes RRMAC's capability to adapt to high traffic load is limited.

Event Based MAC (EB-MAC)

EB-MAC [80] is tailored for event based applications. In such applications, nodes generate traffic when they detect events, but otherwise sleep. When some neighbouring nodes detect an event, they will cause high contention due to simultaneous event reporting. This MAC protocol tries to reduce contention by scheduling packet transmissions for nodes that can sense the event and are within each other transmission range. EB-MAC operates using B-MAC's Clear Channel Assessment (CCA) and Low Power Listening (LPL) techniques. When nodes detect an event, they exchange RTS/CTS packets to form a group and the first node to send the RTS packet becomes the group leader. The group leader builds transmit schedules for

other nodes in its group according to their Received Signal Strength (RSS) of the event detected. The higher the RSS reading, the earlier the slot is given to a node. The schedule is informed to the group members in CTS packets. In order to synchronise the timers of nodes in the group to the leader, EB-MAC uses Flooding Time Synchronisation Protocol (FTSP) [76].

The simulation results in [80] show that EB-MAC improves on B-MAC by 10 times higher throughput with around 50% lower latency when events are detected. However, built on top of B-MAC makes EB-MAC suffer B-MAC's idle listening. Moreover, TDMA schedules are only built for nodes around a detected event, leaves the message passing to the sink in unreliable contention mode.

BurstMAC

BurstMAC [98] utilises TDMA techniques and multiple radio channels for parallel communication to handle correlated traffic burst. It targets low overhead both in idle mode and during correlated traffic bursts. With an assumption that the available number of radio channels n is larger than the maximum number of two-hop nwighbours of a node in the network, BurstMAC uses $n-2$ interference-free channels for data transmissions, one control channel for time synchronisation and one dedicated wake-up channel for network startup. With BurstMAC, each node is assigned a *colour* ID $c \leq n-2$ that is unique within two-hop neighbourhood. This colour ID is used by a node as a channel ID for data transmissions and to schedule broadcasting of control messages on the control channel.

BurstMAC requires a rigid network-wide synchronisation. Every data transmission must be preceded by a request bit from a sender, a synchronisation message by the receiver, a request bit from the sender again in the corresponding colour segment, schedule transmission by the receiver, data by the sender and an acknowledgement by the receiver. This sequence benefits packet bursts because a sender can send a sequence of packets with only a single preamble and a single acknowledgement. However, it incurs high overhead when the traffic is light. BurstMAC supports

dynamic topology changes by letting nodes broadcast information about selected colour IDs in their control messages. A new node can learn the colour sequence and randomly pick a colour ID from the remaining free colours. BurstMAC supports fairness by randomly selecting a sender if there are burst requests by multiple senders. However, in convergecast scenario, the main benefit of BurstMAC's multi channel approach is only the reduction of collision and energy consumption, because its effectiveness is limited by the fact that all traffic has to go up to the sink through a single channel. In the experiment, BurstMAC is reported to achieve less than 1% duty cycle in idle mode and can deliver burst packets five times faster than scheduled MAC protocols.

i-MAC

i-MAC [32] is a MAC protocol developed for manufacturing machines. This protocol is designed to work with repetitive traffic patterns and is adaptive when the traffic loads increase or decrease. i-MAC is a hybrid of TDMA and Frequency-Division Multiple Access (FDMA) approaches. The base station has several channels for simultaneous communication with several sensor nodes within a time slot. i-MAC is topology adaptive because the author assumes that the monitoring area is very dense, where all sensor nodes are within one-hop of each other including the base station. The base station knows the traffic pattern of the network and periodically executes the slot assignment algorithms. The schedules are then disseminated to all sensor nodes.

i-MAC attempts to minimise the transmission latency by reducing the number of time-frequency slots. Therefore, it may assign the same slot to several nodes that have no or low possibility of transmitting at the same time, while sensor nodes that have high possibility of transmitting together are assigned different transmission slots. This approach can reduce the transmission latency when the traffic load is light. However, if two nodes are given the same slot and traffic load increases, transmissions may fail because they keep trying to transmit in the same slot. This scheme

introduces more collisions that can delay data transmissions. When collisions still occur after several consecutive frames, the two nodes randomly choose other time-frequency slots for their next transmissions until new schedules are generated by the base station.

2.3.4 Discussion

Table 2 shows the comparison of the existing MAC protocols that we review in this thesis. We compare all important issues in MAC protocol design for emergency response WSNs, which include the main objectives of the protocols, the ability to adapt to traffic and topology changes, as well as the availability of the design criteria to prioritise high priority packets and to support fairness. A MAC protocol is fair if all nodes have opportunity to access the channel for data transmissions and therefore the sink can receive complete information from all sensor nodes in the network.

Among all existing MAC protocols, only MaxMAC [56] satisfies the objectives for emergency response WSNs, i.e. energy-efficient during light traffic load, has high delivery rate and low latency when the load increases. MaxMAC also has ability to adapt to traffic and topology changes. However, this protocol does not support packet prioritisation and and does not guarantee fairness. When the traffic load is light, MaxMAC behaves like WiseMAC [39] and it changes to pure CSMA when the load is heavy. Both WiseMAC and CSMA do not guarantee fairness because nodes with lots of data dominate the transmissions. Besides MaxMAC, BurstMAC [98] can be utilised for emergency response as it is designed for event-triggered applications with correlated traffic bursts. It has low overhead and high throughput because traffic is handled using multiple radio channels. Even though BurstMAC guarantees fairness, it does not support packet prioritisation.

Judging solely from the ability to adapt to traffic and topology changes, besides MaxMAC and BurstMAC, only Z-MAC [97] and Funneling-MAC [7] have these

Table 2: Comparison of existing MAC protocols

Protocols	Main Objectives	Traffic Adaptability	Topology Adaptability	Packet Priority	Fairness
Contention-based					
S-MAC [123]	↓ energy	medium	good	no	no
T-MAC [114]	↓ energy	medium	good	no	no
B-MAC [88]	↓ energy	medium	good	no	medium
WiseMAC [39]	↓ energy	medium	good	no	no
TA-MAC [48]	↑ delivery, ↓ latency	medium	good	no	no
X-MAC [27]	↓ energy, ↓ latency	medium	good	no	medium
MaxMAC [56]	↓ energy, ↑ delivery, ↓ latency	good	good	no	no
Schedule-based					
TRAMA [94]	↓ energy	medium	good	no	yes
FLAMA [93]	↓ energy	medium	good	no	yes
VTS [38]	bounded latency	medium	good	no	yes
Hybrid					
Z-MAC [97]	↑ throughput	good	good	no	yes
PMAC [127]	↓ energy, ↑ throughput	good	medium	no	yes
Funneling-MAC [7]	↑ throughput	good	good	no	medium
Crankshaft [50]	↓ energy	medium	good	no	medium
RRMAC [64]	↑ delivery, ↓ latency	medium	good	no	yes
EB-MAC [80]	↑ delivery, ↓ latency	medium	good	no	no
BurstMAC [98]	↓ overhead, ↑ throughput	good	good	no	yes
i-MAC [32]	↓ latency	medium	good	no	medium

capabilities. While Z-MAC supports fairness, Funneling-MAC only guarantees fairness in the region closer to the sink. Moreover, both of them do not distinguish high and low priority packets. We identify a gap in the research literature for a MAC protocol that satisfies all of the requirements, i.e. minimises energy consumption when the traffic load is light, has high delivery rate and low latency when the traffic load increases, adapts to traffic fluctuations and topology changes, supports packet prioritisation and has fair packet deliveries in both normal and emergency situations. Our novel MAC protocol in Chapter 4 is designed to satisfy all of these criteria.

2.4 Relay Placement Algorithms

To be able to offer reliable delivery when failures occur, a communication protocol depends on a physical network topology that guarantees alternative routes to the sink are in fact available. Therefore, one key objective in the topology planning

of a WSN is to ensure some measure of robustness. In particular, one standard criterion is to make sure routes to the sink are available for all remaining sensor nodes after the failures of some sensor nodes or radio links. In addition, since there are sometimes data latency requirements, there may be a limit to the path length from sensor to sink. This can be achieved by planning the deployment so that every sensor node in the initial design has disjoint paths with a length constraint to the sink. To ensure that the sensors have sufficient paths, it may be necessary to add a number of additional relay nodes, which do not sense, but only forward data from other nodes.

In network topology planning, sensor nodes, relays and sinks are represented by vertices, and the radio links between them by edges. Two paths are *vertex-disjoint* (respectively *edge-disjoint*) if both of them do not share any vertices (respectively any edges), except the source and the sink. Vertex-disjoint paths are more resilient to failures than edge-disjoint paths [106], because if a source node has k vertex-disjoint paths, it is guaranteed to have a path to the sink after the failure of up to either $k-1$ nodes or $k-1$ radio links. On the other hand, edge-disjoint paths only protect against link failures. Figure 20(a) illustrates a network where the source node s has 2 vertex-disjoint paths to the sink t, while the example in Figure 20(b) shows a network where s has 2 edge-disjoint paths to t, but the paths are not vertex-disjoint. Since we are only interested in vertex-disjoint paths, we will use the term disjoint paths for short throughout this thesis, unless we want to differentiate it from the edge-disjoint ones.

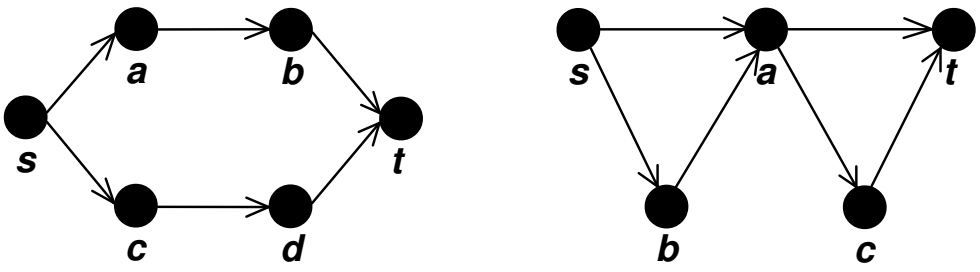


Figure 20: Examples of (a) vertex-disjoint and (b) edge-disjoint paths from the source s to the sink t

Finding several disjoint paths between a source and a sink is motivated by the following advantages [113]:

1. **Improving network reliability and survivability.** The network can use the alternative paths on demand to deliver messages if a path fails or becomes congested and cannot provide the required quality of service. The availability of k disjoint paths is able to tolerate failure of up to $k-1$ nodes.

2. **Providing multi-path routing capability.** Multi-path routing protocols can use all routes simultaneously to minimise latency or to provide redundancy in data transmission. Multi-path routing makes failure much less likely as all disjoint paths must become disconnected to interrupt the transmission.

Installing additional relay nodes, to ensure that sensor nodes have sufficient paths, comes at a cost that includes not just the hardware purchase but more significantly the installation and ongoing maintenance, thus motivating solutions that minimise the number of additional relay nodes. The relay placement problem for WSNs is concerned with deploying a minimum number of relay nodes into the networks to guarantee certain connectivity and survivability requirements. A classification scheme for relay placement problems is shown in Figure 21.

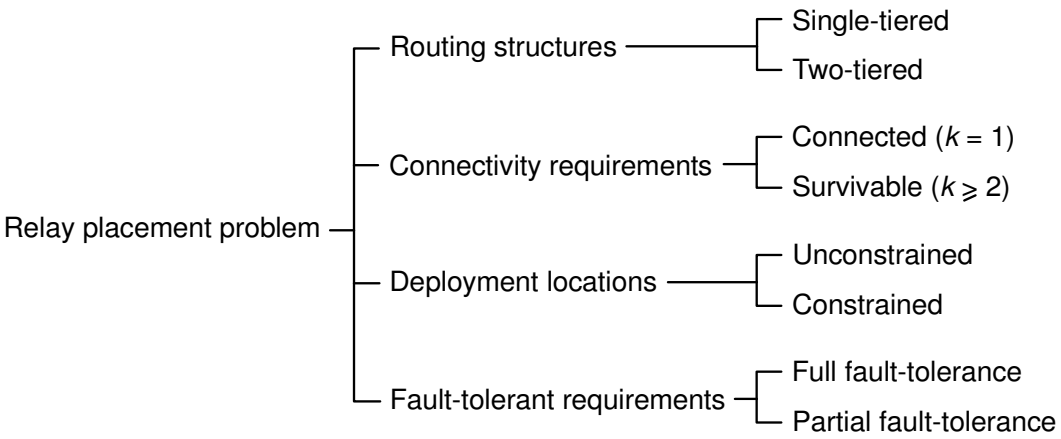


Figure 21: Relay placement problem classification according to [81] and [90]

Misra *et al.* [81] classify the relay placement problems based on the routing structures, the connectivity requirements and the deployment locations. Based on the routing structures, relay placement problems are categorised into *single-tiered* and *two-tiered*. In single-tiered, a sensor node also becomes a relay node to forward packets received from other nodes. The two-tiered network is a cluster-based network, where a sensor node only forwards its own data to a cluster head. Based on connectivity requirements, the problems are categorised into *connected* and *survivable*. In the connected relay placement, a small number of relay nodes is deployed to guarantee that the sensor nodes and the sinks or base stations are connected. In survivable relay placement, the relay nodes are placed to guarantee k-connectivity, where $k \geq 2$. Based on the deployment locations, the problems are divided into *unconstrained* and *constrained*. In the unconstrained relay placement, relay nodes can be placed anywhere. However, in practice, there are some limitations on the possible locations to deploy relay nodes. For example, relays cannot be placed at physical obstacles. In the constrained relay placement, relay nodes can only be deployed at a subset of candidate locations.

The relay placement problem is also classified based on the fault-tolerant requirements, i.e. *full fault-tolerance* and *partial fault-tolerance* [90, 51]. Full fault-tolerance aims to deploy relay nodes in a network to establish k-connectivity between every pair of sensor nodes (original nodes) and relay nodes (additional nodes). Partial fault-tolerance aims to deploy relay nodes to establish k-connectivity only between every pair of sensor nodes as the original nodes. Full fault-tolerance has two properties [26]:

1. the network requires k node failures to disconnect it, and

2. there exist at least k vertex-disjoint paths between every pair of nodes in the network, not just between every node to a dedicated sink.

However, in some cases, partial fault-tolerance is preferable [90], because:

1. only the original nodes serve a useful purpose, the additional nodes merely provide additional connectivity,

2. partial k-connectivity is more economical than the full k-connectivity, because it requires fewer deployed relays.

The relay node deployment problem has long been acknowledged as significant. In this section, we discuss the range of existing algorithms to deploy relay nodes for fault-tolerance. We categorise the reviewed algorithms based on the routing structures, i.e. single-tiered and two-tiered relay placement problems. Recall that in the two-tiered cases, sensor nodes are only within one hop from the relays that serve as cluster heads. Hence, the objective of the partial fault-tolerance is not to provide alternative paths for sensor nodes, but for relay nodes. The reviewed relay placement algorithms for WSNs are summarised in Table 3.

2.4.1 Single-tiered Relay Placement Problem

k-Connectivity-Repair

Bredin *et al.* [26] develop k-Connectivity-Repair as a centralised greedy algorithm and its distributed version for the single-tiered unconstrained full fault-tolerant relay placement problem to guarantee vertex k-connectivity. They assume that relay nodes have the same transmission range as sensor nodes and the range is normalised to one. The algorithm firstly computes a weighted complete graph, where the weight of an edge is one less than the Euclidean distance between a pair of sensors. The edge's weight represents the number of additional relays required to connect two sensors by a straight path. After that, this algorithm finds an approximate minimum-weight vertex k-connected subgraph by repeatedly adding edges in increasing order of weight until the subgraph is k-connected. If the subgraph is already k-connected, it repeatedly attempts to remove edges in decreasing order of weight, but putting the edge back if it is important for k-connectivity. Finally, it

Table 3: Summary of existing relay placement algorithms

Algorithms	Summary
Single-tiered	
k-Connectivity-Repair [26]	From a weighted complete graph, finds a minimum-weight vertex k-connected subgraph by adding edges in increasing weight. For each edge, deploys k relays every transmission range distance and $k-1$ relays at endpoints of the edge.
Partial k-Connectivity-Repair [90]	Similar to k-Connectivity-Repair, but only places one relay every transmission range distance and none at endpoints.
Connectivity-First [51]	From a weighted complete graph, finds a minimum k-connected spanning graph by adding edges that have the highest contribution to connectivity and the least weight.
Redundant Router Placement [6]	Uses Ford-Fulkerson to count the number of paths from sensor to sink. It adds paths by placing relays start from the furthest sensor from the sink.
1tFTP and 2tFTP [126]	1tFTP constructs a complete graph, finds a 2-connected spanning subgraph and steinerises the edges. 2tFTP finds the fewest relays as cluster heads, connects them using the Steiner minimum tree and duplicates each relay found.
RNP_C and RNP_S [81]	Assign edges' weight as the number of candidate relays they are incident with. RNP_C computes a low weight connected subgraph. RNP_S computes a low weight 2-connected subgraph. Relays are deployed at the candidate locations that appear in the subgraph.
Two-tiered	
2CRNDC [52]	In each iteration, it selects a relay that can cover as many sensors, which are not covered by two relays, as possible. Then, it selects some relays that can make the previously selected relay have two disjoint paths.
CRNSC and 2CRNDC [110]	Divide region into cells, find possible positions to deploy relays, find a solution to cover ($k=1$) or double cover ($k=2$) sensors in each cell using exhaustive search, then add extra relays if needed.
MRP-1 and MRP-2 [70]	MRP-1 finds the fewest relays that can cover all sensors and connects them using the Steiner minimum tree. MRP-2 adds three relays in the transmission range's circle of each relay found in MRP-1.
k-Vertex Connectivity [61]	From a complete graph of cluster heads, calculates edges' weight, finds a minimum cost vertex k-connected spanning subgraph, and deploys relays along the subgraph's edges.

places clusters of k relays along each edge every one unit distance and $k-1$ relays at both endpoints of the edge.

The simulation results show that the distributed version of the algorithm nearly achieves the same number of required additional relays as the centralised greedy version. Moreover, compared to the random repair algorithm, where relays are scattered randomly until the k-connectivity is achieved, the two versions of k-Connectivity-Repair only require one seventh of the random repair cost to restore graph 3-connectivity.

Partial k-Connectivity-Repair

Pu *et al.* [90] propose Partial k-Connectivity-Repair by modifying the k-Connectivity-Repair algorithm by Bredin *et al.* [26] to guarantee only partial fault-tolerance. Partial k-Connectivity-Repair follows the same procedure as k-Connectivity-Repair to compute a weighted complete graph and to find a minimum-weight vertex k-connected subgraph. After that, instead of placing clusters of k relays along each edge every one unit distance and $k-1$ relays at both endpoints of the edge for full fault-tolerance, the proposed modification for partial fault-tolerance only deploys one relay every transmission range distance and none at the endpoints of each edge.

Connectivity-First

Han *et al.* [51] develop algorithms for the single-tiered unconstrained partial and full fault-tolerant relay placement problem for $k \geq 1$. They assume heterogenous WSNs, where sensors have different transmission radii, while relays use the same transmission radius. This asymmetric communication links together with the level of desired fault-tolerance divide the problem into four categories: one-way and two-way partial fault-tolerant, and one-way and two-way full fault-tolerant relay placement. The algorithms firstly calculate the weight of additional edges between each pair of sensors in a complete graph. The weight determines how many relays

needed along a straight line between two sensors. It is calculated by dividing the Euclidean distance of the two sensors by the relay's transmission radius.

A greedy heuristic algorithm called Connectivity-First is then proposed to find the minimum k-connected spanning graph. It adds edges that can best help improving the connectivity until the graph becomes k-connected. An additional edge is selected because it has the highest contribution to the connectivity and has the least weight, i.e. the number of relays. The connectivity is checked using a maximum network-flow-based checking algorithm [86] as is used in [95]. When the graph is k-connected, the algorithm tries to remove redundant edges in decreasing order of weight as long as the removal does not break the k-connectivity. Finally, a number of relays is deployed along the selected additional edges. The results show that the algorithm by Bredin *et al.* [26] is more efficient for partial fault-tolerance, while Connectivity-First is more efficient for full fault-tolerance in terms of the number of relays that needs to be added to the network.

Redundant Router Placement

Ahlberg *et al.* [6] study the problem of single-tiered unconstrained partial fault-tolerant relay placement for $k=1$ and $k\geq 2$. In the non-redundant relay placement ($k=1$), they propose three algorithms:

1. *Trivial Router Placement* simply deploys relays on a straight line from each and every sensor to the sink.

2. *Trivial Placement Reusing Routers* sorts the sensors according to their distances to the sinks, connects the closest sensor to its sink by deploying relays on a straight line, and then connects the next closest sensor to the closest deployed relays or to the sink.

3. *Cluster Router Placement* groups nearby and connected sensors into a cluster and uses the Trivial Router Placement algorithm to connect clusters, instead of connecting each sensor separately.

For the redundant relay placement ($k \geq 2$), firstly the algorithm counts the number of available paths from each sensor to the sink using the Ford-Fulkerson maximum flow algorithm (see Appendix A.2 for the pseudocode). If the number of available paths is not sufficient, the algorithm places redundant relays start from the furthest sensor from the sink.

Further, to reduce the number of deployed relays, Ahlberg *et al.* suggest two optimisation techniques:

1. For the non-redundant placement ($k = 1$), all sensors are reconnected to the relay that has the shortest path to the sink. Relays with only connection to another relay are removed.

2. For the redundant placement ($k \geq 2$), each relay is temporarily removed and the number of available paths are recalculated, but placing it back if necessary.

Single-tiered and Two-tiered Fault-Tolerant Relay Placement (1tFTP and 2tFTP)

Zhang *et al.* [126] study the single-tiered and two-tiered unconstrained partial fault-tolerant relay placement problem for k-connectivity, where $k = 2$. Relay nodes are assumed to have larger transmission range than sensor nodes. The network may also have base stations. The proposed algorithms are:

1. *Single-tiered Fault-Tolerant Relay Placement* (1tFTP). It constructs a complete graph, computes a 2-connected spanning subgraph and steinerises the edges of the subgraph. The *steinerisation* process calculates edges' weight by dividing the Euclidean distance of any two vertices by the relay's transmission radius. For each edge, a number of relays is deployed along the straight line.

2. *Two-tiered Fault-Tolerant Relay Placement* (2tFTP). It uses the Two-tiered Relay Node Placement (2tRNP) algorithm that is developed for 1-connectivity

proposed by Lloyd and Xue [71]. 2tRNP finds the minimum number of relays that can cover all sensors into one-hop clusters. Relays in all clusters are then connected by paths of additional relays. For this, 2tRNP finds the Steiner minimum tree with minimum number of Steiner points. 2tFTP then duplicates each of the relays found by 2tRNP.

1tFTP and 2tFTP are compared to two heuristics, 1tTSP and 2tTSP, that may produce close to optimal solutions. 1tTSP and 2tTSP compute a Traveling Salesman (TSP) tour of the graph and steinerise the edges of the tour to deploy relays. The simulation results show that in all cases with varied network density, the numbers of relays required by 1tFTP and 2tFTP are no more than 1.5 times the numbers of relays required by 1tTSP and 2tTSP.

Connected and Survivable Relay Node Placement ($\mathbf{RNP_C}$ and $\mathbf{RNP_S}$)

Misra *et al.* [81] study the single-tiered constrained partial fault-tolerant relay placement problem for both the connectivity ($k=1$) and the survivability ($k=2$) requirements. They assume that the transmission range of sensor nodes is smaller than the transmission range of relay nodes. Misra *et al.* propose:

1. *Connected Relay Node Placement* ($\mathrm{RNP_C}$) for $k=1$. It firstly constructs the communication graph for sensors, base stations and relays' candidate locations. Then, it assigns edges' weight as the number of candidate relays they are incident with. Finally, the low weight tree subgraph is computed, from which the locations to place relays are identified. The unconstrained version of $\mathrm{RNP_C}$ is Single-tiered Relay Node Placement (1tRNP) studied by Lloyd and Xue [71], where there is no restriction on the locations of the relays.

2. *Survivable Relay Node Placement* ($\mathrm{RNP_S}$) for $k = 2$. The algorithm constructs the communication graph and assigns edges' weight too. It then assigns connectivity requirements between every pair of vertices in the following way: $c(v, w) = 2$ if neither v, nor w is the candidate for relay. Otherwise,

$c(v, w) = 0$. Then, the low weight 2-connected subgraph that meets the connectivity requirements is computed. Relay s' candidate locations that appear in the subgraph are the positions to deploy additional relays.

In the simulation, RNP_C and RNP_S are compared to simulated annealing. The results show that they are able to produce almost the same numbers of relays as the results obtained by simulated annealing. Simulated annealing has 10 times longer running time, but only finds slightly better solutions in a few cases.

2.4.2 Two-tiered Relay Placement Problem

2-Connected Relay Node Double Cover (2CRNDC)

Hao *et al.* [52] propose an algorithm to solve the two-tiered constrained partial fault-tolerant relay placement problem for 2-connectivity. Under an assumption that the distributed sensor nodes are already 2-connected, they want each sensor node to be able to communicate with at least two relay nodes and the network of the relays is 2-connected. They also assume that the relay nodes' transmission range is at least twice the transmission range of sensor nodes. In each iteration, the algorithm selects one relay from the set of candidate positions that can best cover as many sensor nodes, which are not covered by two relays, as possible. Then, it selects some relays from the set of candidate positions that can make the previously selected relay have two disjoint paths and become 2-connected. The algorithm proceeds until all sensors in the network are covered by at least two relays.

Connected Relay Node Single Cover (CRNSC) and 2-Connected Relay Node Double Cover (2CRNDC)

Tang *et al.* [110] study the problem of two-tiered unconstrained partial fault-tolerant relay placement for $k = 1$ and $k = 2$. Under an assumption that the transmission range of relay nodes is four times the range of sensor nodes, they propose:

1. *Connected Relay Node Single Cover* (CRNSC). It requires that each sensor node to be covered by at least one relay node, and that the set of relay nodes is connected.

2. *2-Connected Relay Node Double Cover* (2CRNDC). It requires that each sensor node to be covered by at least two relay nodes and that the network induced by the relay nodes is 2-connected.

The main ideas of the algorithms are:

1. Divide the region into small cells of size $l.2r$, where l is an integer partition factor and r is the transmission range of sensor nodes.

2. For each cell, find all possible positions to deploy relays. Possible positions are the intersections of sensors' transmission circles of radius r. If a possible position is outside of a cell, it is replaced with the closest point on the border of the cell.

3. Without considering the connectivity, find the optimal solution to cover ($k=1$) or double cover ($k=2$) the sensor nodes within each cell using exhaustive search.

4. Make the network of relays connected ($k=1$) or 2-connected ($k=2$) by adding extra relays at some specific locations if necessary.

Minimum Relay-Node Placement for 1 and 2-Connectivity (MRP-1 and MRP-2)

Liu *et al.* [70] address the two-tiered unconstrained full fault-tolerant relay placement problem for $k=1$ and $k=2$. In the hierarchical network, relay nodes act as cluster heads and are connected with each other to perform data forwarding task. The proposed algorithms are:

1. *Minimum Relay-Node Placement for 1-connectivity* (MRP-1). The first step is finding the minimum number of relay nodes that can cover all sensor nodes. The network of relays may not be connected if the distance between them is larger than the transmission range. Therefore, more relays are needed. The second step is constructing Steiner tree to connect the relays such that the number of Steiner points, in this case the additional relays, is minimised.

2. *Minimum Relay-Node Placement for 2-connectivity* (MRP-2). To achieve 2-connectivity, MRP-2 adds three additional relay nodes in the transmission range's circle of each relay found in MRP-1.

These two algorithms can be utilised to the cases where the transmission ranges of sensor nodes and relay nodes are either the same or different.

k-Vertex Connectivity

Kashyap *et al.* [61] give algorithms for the two-tiered unconstrained and constrained partial fault-tolerant relay placement problem for edge and vertex k-connectivity, where $k \geq 2$. They assume a hierarchical network, where sensors forward data to cluster heads. Therefore, the network should have vertex-disjoint (or edge-disjoint) paths between each pair of cluster heads. Relay nodes are assumed to have the same communication capabilities as the cluster heads and the range is normalised to one. The algorithm for vertex k-connectivity starts by constructing a complete graph of cluster heads and calculating the edges' weight as the number of relays needed. The weight is calculated from the edge's length minus one. Then, the minimum cost vertex k-connected spanning subgraph is sought. After that, relays are placed along the additional edges of the resulting subgraph. Finally, the algorithm tries to remove relays, which are sorted arbitrarily, one by one by still preserving the vertex k-connectivity. The resulting graph is vertex k-connected.

Table 4: Comparison of existing relay placement algorithms

Algorithms	k	R **vs** r	**Routing**	**Deployment Locations**	**Fault-Tolerance**
Single-tiered					
k-Connectivity-Repair [26]	≥ 1	$R=r$	1-tiered	unconstrained	full
Partial k-Connectivity-Repair [90]	≥ 1	$R=r$	1-tiered	unconstrained	partial
Connectivity-First [51]	≥ 1	$R\geq r$	1-tiered	unconstrained	full/partial
Redundant Router Placement [6]	≥ 1	$R\geq 2r$	1-tiered	unconstrained	partial
1tFTP and 2tFTP [126]	2	$R\geq r$	1/2-tiered	unconstrained	partial
RNP_C and RNP_S [81]	1, 2	$R\geq r$	1-tiered	constrained	partial
Two-tiered					
2CRNDC [52]	2	$R\geq 2r$	2-tiered	constrained	partial
CRNSC and 2CRNDC [110]	1, 2	$R\geq 4r$	2-tiered	unconstrained	partial
MRP-1 and MRP-2 [70]	1, 2	$R=r$, $R\neq r$	2-tiered	unconstrained	full
k-Vertex Connectivity [61]	≥ 2	$R=r$	2-tiered	un/constrained	partial

2.4.3 Discussion

We show the comparisons of the reviewed relay placement algorithms for WSNs in Table 4. We compare the algorithms based on the connectivity requirements (k), the assumption made on the transmission ranges, i.e. R and r denote the transmission ranges of relay nodes and sensor nodes, respectively, the routing structures, the deployment locations, and the fault-tolerant requirements. Recall that for $k = 1$, the algorithm only guarantees that the network is connected. If $k \geq 2$, it guarantees survivability. Relay nodes can only be placed at a subset of candidate locations in the constrained deployment, but can be placed anywhere if the deployment locations are unconstrained.

In this thesis, we assume that an initial WSN topology is connected and additional relays may be required for fault-tolerance. Even though relays may die during the network operation, we only protect the network against sensor node failures because relays only provide additional connectivity to improve the network reliability and survivability. Hence, we focus our research on the partial fault-tolerant relay placement. Furthermore, since in practice relays cannot be placed anywhere inside the monitoring region, we only consider the constrained relay placement. Based

on the state-of-the-art relay placement algorithms that we reviewed in this thesis, we identify two research opportunities. Firstly, there is a gap in the research literature for a relay placement algorithm for the single-tiered, constrained partial fault-tolerant relay placement problem for $k \geq 2$. The closest approach is RNP_S by Misra *et al.* [81], but it is designed only for $k = 2$. Other algorithms, namely Partial k-Connectivity-Repair [90], Connectivity-First [51] and Redundant Router Placement [6], are designed for unconstrained deployment locations. Secondly, there is a research opportunity for a relay placement algorithm that takes into account a path length constraint, since all reviewed algorithms do not consider this issue. We will discuss a new solution for the single-tiered, constrained partial fault-tolerant relay placement problem for $k \geq 2$ disjoint paths with a length constraint in Chapter 5.

2.5 Disjoint Path Algorithms

Offline algorithms to discover k shortest vertex-disjoint and edge-disjoint paths in existing networks are well studied in the literature. A maximum flow algorithm, such as the Ford-Fulkerson algorithm [43], can be used to find edge-disjoint paths [65] in a graph. Since in this thesis we consider both node and link failures, we will only focus our research on vertex-disjoint paths and omit the discussion about edge-disjoint paths as they can only protect a network against link failures. We present below the existing algorithms to compute the shortest vertex-disjoint paths.

Fast Pathfinding, Maximum Paths and Refined Maximum Paths

Torrieri [113] presents three algorithms to calculate a set of short disjoint paths, which do not exceed the longest acceptable path length, between a source and a sink. In each iteration, all three algorithms select the shortest path, remove the intermediate vertices in the path from further use by zeroing the rows and the columns of the intermediate vertices in the adjacency matrix and then select the

next shortest path using only the remaining vertices. The three algorithms proposed by Torrieri are:

1. *Fast Pathfinding*. This is the simplest approximate algorithm, which executes only the first step in the construction of the optimal set. In this algorithm, if two or more remaining paths of length l are the shortest, one of them is chosen arbitrarily.

2. *Maximum Paths*. This approximate algorithm executes the first two steps in the construction of the optimal set. That is, if two or more remaining paths of length l are the shortest and they exclude the fewest other paths of length l, then one of the remaining paths is chosen arbitrarily.

3. *Refined Maximum Paths*, which constructs an optimal set of short disjoint paths without approximation. This is similar to Maximum Paths except when two or more paths have the same length and exclude the same number of other paths of that length, two or more parallel computations occur.

Vertex-Disjoint Shortest Pair of Paths

The disjoint shortest pair of paths algorithm proposed by Bhandari [19] requires two runs of a modified Dijkstra algorithm to find two shortest paths between a source and a sink. In this paper, Dijkstra's algorithm is slightly modified to handle negative directed edges. The main idea is to exclude all possible paths between the source and the sink that intersect with the first shortest path found during the search for the second shortest path. Exclusion of such path is achieved by vertex-splitting along the first shortest path found. Bhandari's algorithm begins by finding the first shortest path for a pair of vertices under consideration using the modified Dijkstra algorithm. The graph is then modified by:

1. replacing edges on the shortest path by negative directed edges toward the source,

2. splitting vertices on the shortest path, joining them by zero weighted directed edges toward the source, and

3. replacing edges connected to vertices on the shortest path by two oppositely directed edges of the original weight.

After that, the modified Dijkstra algorithm is run again on the modified graph. The original graph is then restored and the overlapping edges of the two paths found are removed to obtain the shortest pair of disjoint paths. For the shortest k disjoint paths problem, where $k > 2$, the algorithm is performed iteratively to obtain more disjoint paths in a given network graph, provided such paths exist.

Localised Algorithm for Finding Node-Disjoint Paths (LAND)

Hou and Shi [55] propose LAND, a localised algorithm to find the shortest disjoint paths from every deployed sensor node to a sink and also to provide localised path restoration. The sink starts the algorithm by sending a message to its direct neighbours. The message that is flooded throughout the network contains the information about the shortest path length and the path identifier, which is the identifier of a sink's neighbour. Upon receiving the message, each sensor node updates its routing information if the message contains shorter path length for a specific path identifier. When a node updates its routing information, it sends an update message to its neighbours. In addition to finding disjoint paths, the path restoration mechanism works as follows: a neighbour of an exhausted node will broadcast a request message and the receivers will initiate the sending of the shortest path message to reconstruct the path.

2.6 Centrality and Alternative Path Centrality

Providing k-connectivity to the whole network [26, 51] so each sensor node has k disjoint paths [19] is costly because it may require the addition of an excessive

amount of relay nodes. Therefore, as other alternatives to k disjoint paths, the concept of *maximally (partial) disjoint paths* is introduced in [20] and *non-disjoint paths* is used in [60] to reduce the resource cost. Two paths from a source s to a sink t are maximally disjoint if there are no disjoint paths from s to t and the two paths share a minimum number of common vertices and edges [20]. Two paths are non-disjoint if both of them overlap [60].

Another solution to reduce the cost of relay deployment is by placing relays to provide additional connectivity only around the most important nodes. The importance of a node in network analysis is called its *centrality*. Originally, it is measured by counting the number of the shortest paths passing through a certain node. However, since we are dealing with node failures, we define the importance of a node based on the effect of removing the node from the network. That is, the node is important if its failure would disconnect many other nodes, or cause traffic from many other nodes to be delivered late. In this situation, we need a centrality measurement where the shortest paths are actually bypassing the node, not passing through it.

The use of centrality index to analyse network robustness has been proposed in the literature. Shavitt and Singer [101, 102] present two new centrality measures based on the existence of a non-disjoint backup path if a node fails. This notion of *alternative path centrality* is proposed for mesh networks. So, it calculates the backup paths between every pair of nodes, not only between nodes and sinks. It also does not consider a path length constraint in its calculation, which is an important aspect for data latency requirements.

In the following subsections, we will discuss the concept of centrality and alternative path centrality. We will present a new variation of alternative path centrality, which takes account of a path length constraint, for WSNs with sinks in Chapter 6. We will also discuss a new solution for the relay placement problem using this new centrality.

2.6.1 Centrality

Centrality is a core concept in social network analysis, which was introduced by Bavelas [16] in 1948. A centrality score is originally calculated by counting the number of the shortest paths passing through a node. In its development, there are several centrality indices that are mostly used. In [44], Freeman distinguishes three basic centrality measures:

1. *Degree centrality* of a vertex v is measured by the number of vertices adjacent to v,

$$C_{\mathrm{D}}(v) = |N(v)|$$

2. *Closeness centrality* of a vertex v is an inverse sum of distances from v to all other vertices in the graph,

$$C_{\mathrm{C}}(v) = \frac{1}{\sum_{s \neq v \in V} d(s,v)}$$

3. *Betweenness centrality* of a vertex v is the sum of the probability that v falls on a randomly selected shortest path between all pairs of vertices ($s \neq t \neq v$),

$$C_{\mathrm{B}}(v) = \sum_{s \neq t \neq v \in V} \sum_{t \neq s \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

   where σ_{st} denotes the number of shortest paths between s and t and $\sigma_{st}(v)$ denotes the number of shortest paths between s and t that pass through v other than s and t.

Brandes [25] gives the variants of the shortest path betweenness, which do not only consider the intermediaries, but also the influence of endpoints, distance, edge, group, etc. In vehicular networks, the concept of centrality is used for access-points deployment [62, 63] and discovering link criticality [99]. In WSN, it is used for routing [84] and load balancing [87].

2.6.2 Alternative Path Centrality

The new centrality measures proposed by Shavitt and Singer [101, 102] are *Quality of Backup* (QoB) and *Alternative Path Centrality* (APC). The idea behind these is the failures of nodes with perfect backups do not affect connectivity nor increase the path length in the network. QoB is a measure of path rerouting from a vertex's direct parents to its direct children. QoB of a vertex v is

$$\rho(v) = \frac{\sum_{u \in \pi_v} \sum_{w \in C_v} \frac{1}{\max\{d_v(u,w)-1,1\}}}{|\pi_v| \cdot |C_v|}$$

where π_v is a set of v's direct parents and C_v is a set of v's direct children. $\rho(v)=1$ if v has perfect backups and $\rho(v)=0$ if v has no backup.

APC is the difference between vertices' topological centrality before and after a vertex fails. The topological centrality of a vertex $u \in V$, denoted $\chi(u)$, depends on the number of vertices connected to u and their distances from u,

$$\chi(u) = \sum_{w \in V \setminus \{u\}} \frac{1}{d(u,w)}$$

Therefore, $0 \leq \chi(u) \leq |V|-1; \forall u \in V$. The APC value of a vertex v is

$$\varphi(v) = \sum_{u \in V \setminus \{v\}} \chi(u) - \sum_{u \in V \setminus \{v\}} \chi_v(u)$$

where χ_v denotes the centrality values calculated using alternative paths which bypass v. Although QoB and APC indices include connectivity information of a network, neither of them can be used to identify which node failures would cause the network to be disconnected. Moreover, they do not consider a length-bound in the calculation.

2.7 Multiple Sink Placement Algorithms

While a relay node has similar resources to a sensor node except the sensing capability, a sink is usually powered, has large storage capacity and has WiFi/ethernet

backhaul. Therefore, the cost of a sink is assumed to be more expensive than a relay node. In a traditional WSN, there is only one single static sink (or a base station) that gathers data from sensor nodes in multi-hop communication, performs data processing and reports it to the end-users. This single sink scenario has several drawbacks, such as the energy hole problem and poor scalability. Moreover, the reliance on one device is also a drawback, because any malfunction of that device disconnects everything.

Having only one sink results in a many-to-one (convergecast) traffic pattern. It means sensor nodes closer to the sink must relay traffic from farther nodes that cannot reach the sink directly. This creates an energy hole around the sink, where nodes near the sink fail quickly because of energy depletion. An energy hole partitions the network and means that other sensor nodes are unable to reach the sink. Moreover, having only one sink that serves the entire network is not scalable since a WSN may consist of hundreds of nodes. This results in low quality of service in the network, such as low throughput and high data delivery latency. In addition, even though a sink has more resources than a sensor node, this electronic device may fail too and is a single point of failure that disconnects the entire network. To mitigate these problems, it is necessary to deploy more than one sink in the monitoring region.

Multiple sink deployment has been extensively studied in the WSN literature. Similar to sink deployment in WSN, many algorithms have also been developed to find the optimal positions of routers or gateways in wireless mesh networks, access point deployment in wireless neighbourhood networks, and core node placement in optical networks, all of which will be discussed in this section. Generally, these algorithms have different objectives from one another. Their objectives, such as minimising energy consumption to prolong the network lifetime, maximising throughput, minimising latency, or supporting fault-tolerance, influence the algorithm designs. To simplify the discussion, we will refer to routers, gateways, access points, core node, cluster heads, data sinks and base stations as sinks, while other nodes, such as mesh

nodes, clients and sensors as nodes. We categorise the range of existing multiple sink placement algorithms based on the number of sinks, whether it is given as an input parameter (fixed) or becomes the function that the algorithms try to minimise. Table 5 presents the summary of our reviewed algorithms. A brief discussion will follow later at the end of this section.

2.7.1 Minimise the Number of Sinks

Heuristic Opt Multisink Place (HOMP)

Xu and Liang [122] propose *Heuristic Opt Multisink Place* (HOMP), a heuristic algorithm to place an optimal number of sinks to prolong the network lifetime. The algorithm consists of two sub-problems: finding the optimal number of sinks from a set of candidate locations so each node's hop count to the sink is no longer than a hop count bound, and building a load-balanced tree-based routing protocol for data collection to maximise the network lifetime. To find the optimal number of sinks, HOMP iteratively selects a sink such that it covers as many nodes, which are within the hop count bound from the sink, as possible. The algorithm terminates when all nodes are covered by the selected sinks. Then, for each cluster, a load-balanced routing tree rooted at the sink is built by minimising the maximum number of descendants of the sink's direct children. Simulation results show that HOMP achieves 13% longer network lifetime compared to the breadth first search tree-based heuristic.

2-Connectivity

Ivanov *et al.* [58] present an algorithm to deploy a minimum number of sinks in a wireless multi-hop backbone to provide fault-tolerance when one sink fails or one link fails (2-connectivity). In graph theory, the minimum degree is necessary but not sufficient condition for k-connectivity. The proposed algorithm starts by checking each node's degree. Then, it performs connectivity testing. If the graph is

Table 5: Summary of existing multiple sink placement algorithms

Algorithms	Summary
Minimise the Number of Sinks	
HOMP [122]	Iteratively selects a sink that covers as many nodes, which are within a hop count bound from the sink, as possible.
2-Connectivity [58]	Identifies 2-connected components and articulation points, then deploys additional sinks to achieve 2-connectivity.
Greedy Placement [91]	Iteratively selects a sink to maximise the flow demands in conjunction with the previously chosen sinks.
Negative Selection [120]	Decides which candidate sinks will be eliminated from further consideration.
OPEN/CLOSE [89]	Lists sinks in a decreasing order of capacities, selects a minimum number of sinks from the top of the list, forms clusters, and updates the solution recursively with lower capacity sinks to reduce cost.
Iter. Greedy DS [17]	Divides a network into a minimum number of clusters with bounded radius by selecting sinks greedily. Each cluster is then divided into sub-clusters if either relay load or cluster size constraints are violated.
Weighted Recursive [14]	Greedily selects high degree nodes as sinks and builds spanning trees. In each iteration with increasing hop count, greedily reselects nodes that can cover as many nodes as possible as sinks.
Incr. Clustering [111]	Uses R-step transitive closure to identify sinks.
MSPOP [85]	Deploys sinks one by one while evaluating the network lifetime. Identifies sink positions using k-means clustering.
Fixed Number of Sinks	
BSL [85]	Uses k-means clustering to find sink positions.
CBS [28]	Uses a variant of k-means clustering to form overlapping clusters. A node joins the clusters of the two nearest sinks.
Cluster Balancing [73]	Finds the optimal location of a sink such that the total shortest hop distance of the cluster is minimised.
MBCP [103]	Partitions a network into connected sub-networks of equal size and places a sink randomly in each sub-network.
Global, 1hop [115]	Sinks form clusters by grouping close-by nodes, then find the centroid of the clusters as their new positions.
DECOMP [82]	Optimises node-sink connections by finding the minimum power of each node to p sinks and optimise each sink's location by moving it to the centre of mass of all nodes connecting to it.
COLA [8]	Divides a region into cells, places a sink at the centre of a cell, forms clusters, and reposition sinks to minimise latency.
GAHO, GADO [124]	Uses genetic algorithms to place each sink at or close to the geometric centroid of all nodes in the same cluster.
PMP [72]	Finds the best position for one sink by moving it across all candidate positions, places all sinks at that position, then tries to find the best positions for them one by one by keeping other sinks untouched.
Greedy, Local Search [22]	The greedy algorithm selects sinks one by one to improve the data rate. Local search starts with a random placement of sinks and relocates them to improve the data rate.
MTWP [128]	Iteratively selects a sink with the highest traffic-flow weight.

2-connected, the algorithm stops. Otherwise, it performs an incremental correction by firstly identifying 2-connected components and articulation points shared between the 2-connected components. An articulation point is a node whose removal disconnects a graph. Finally, the algorithm deploys additional sinks to make the graph 2-connected.

Greedy Placement

Qiu *et al.* [91] study the problem of minimising the number of sink placements to maximise the bandwidth utilisation in three scenarios: ideal link model, bounded hop count model and smooth throughput degradation model. In the ideal link model, the throughput of each link along a path is assumed to be one. In the bounded hop count model, the throughput is one if the path length is not more than a hop count threshold, otherwise it is zero. In the smooth throughput degradation model, the amount of throughput on an link along a path of length l is $\frac{1}{l}$. Given a capacity constraint for each link, node and sink, the authors propose greedy placement algorithms for these three models, where a sink is iteratively picked that maximises the total flow demands satisfied in conjunction with the sinks chosen in the previous iterations. The greedy algorithms utilise the Ford-Fulkerson network flow algorithm to calculate the total flow demands. In the simulation, it is observed that an increase in the communication radius results in fewer number of sinks chosen to satisfy the flow demands. The simulation results also show that the three proposed greedy algorithms performs very close to the optimal solutions and achieves 2 to 10 times as few sinks compared to the random deployment.

Negative Selection Statistically-tuned

In [120], Wong *et al.* propose the *Negative Selection Statistically-tuned* heuristic algorithm to find locations of a minimum number of sinks to minimise communication delay and hop count from nodes to the nearest sinks. Unlike other heuristics,

at each step the proposed heuristic algorithm decides which of the candidate sinks will be eliminated from further consideration based on the weighted sum of four components: lower bounds, average node difficulty for coverage, greedy heuristic and randomised greedy heuristic. The lower bound is calculated as the sum of the inverse of the maximum number of nodes covered by a sink, i.e. nodes to which the sink can communicate. The difficulty to cover a node is the inverse of the square of the number of sinks that currently cover that node. The greedy heuristic selects the sink with the largest number of covered nodes. The randomised version of the greedy algorithm probabilistically selects a sink based on the coverage by the sink.

OPEN/CLOSE

Prasad and Wu [89] investigate the problem of deploying a minimum number of sinks to minimise the network installation cost and balance the traffic by taking into account the sinks' maximum capacity to handle traffic. The network installation cost takes into account the total number of hops from each node to the sinks and the cost of choosing the sinks. Note that a higher capacity sink has a higher cost. The authors propose a heuristic algorithm called *OPEN/CLOSE*. The heuristic algorithm picks the initial solution by listing all potential sinks in a decreasing order of bandwidth capacities and picking a minimum number of sinks from the top of the list whose combined capacities satisfies the total bandwidth requirement of the network. Clusters are formed where each node finds the shortest path to the nearest sink that still has remaining bandwidth capacity. The initial solution consists of high capacity sinks, so the total cost is high. The algorithm then recursively updates the solution by replacing one sink at a time with a few other sinks from the candidate locations to reduce the cost but still satisfy the network's bandwidth requirement. The simulation results show that OPEN/CLOSE deploys 20% lower cost sinks than a greedy approach.

Iterative Greedy Dominating Set

In [17], Bejerano proposes the *Iterative Greedy Dominating Set* algorithm to partition the network into a minimum number of disjoint clusters and find the position of sinks. The formed clusters must satisfy several constraints: the delay bound (cluster radius), the relay load (a node can serve as relay only for a limited number of other nodes), and the bandwidth requirement of all nodes in the cluster. In the first step, the algorithm divides the network into a minimum number of clusters with bounded radius by selecting the sinks greedily. In each iteration, a node which can cover the maximum number of other nodes in its radius-bounded neighbourhood is selected as a sink. Then, clusters are formed and the shortest path trees are built, where each node selects the nearest sink to join. If a tree violates the bandwidth or relay load constraints, it is divided into smaller subtrees that meets all of the requirements. To further reduce the maximum relay load, a heuristic algorithm is employed to reposition the sinks by considering all nodes in the clusters that can serve as the root of the tree.

Weighted Recursive

Aoun *et al.* [14] propose *Weighted Recursive*, a recursive greedy algorithm to deploy a minimum number of sinks such that the Quality of Service (QoS) requirements are satisfied. The QoS constraints concerned here are the delay bound (cluster radius is at most R hops), the relay traffic (a node can only relay at most L other nodes' traffic), and the sink capacity (cluster size is at most S nodes). The algorithm recursively computes minimum weighted dominating sets. The weight of a node v is not simply the number of nodes it covers, but a weighted sum, i.e. a node farther from v will have a lower contribution to the total weight of v. The algorithm starts by greedily selecting nodes with high degree as sinks and building spanning trees. In each iteration with increasing hop count, it greedily reselecting nodes that can cover as many nodes as possible as sinks and reconstructing the spanning trees. When

reconstructing the spanning trees, the relay traffic and the sink capacity bounds are checked to guarantee QoS. The algorithm stops when each cluster's radius is at most R hops. Simulation results show that the weighted recursive outperforms the iterative greedy dominating set [17] and the augmenting placement by 50% fewer sinks. The augmenting placement is similar to the iterative greedy placement, but it does not make greedy decisions for the next sink placement. Any placement providing subsequent coverage to uncovered nodes is considered.

Incremental Clustering

Tang [111] proposes the *Incremental Clustering* algorithm that incrementally identifies sinks and assigns nodes to the identified sinks. The algorithm is designed to satisfy three QoS constraints, namely the delay bound (communication is at most R hops), the relay load (a node can only relay at most L other nodes' traffic), and the sink capacity (a sink can only serve at most S nodes). The algorithm firstly builds the R-step transitive closure from the graph representation of the network. An edge in the R-step transitive closure represents a path in the original graph that has length less than or equal to R. The i^{th} row of the R-step transitive closure is a cluster representing a set of nodes that can be covered by the i^{th} node. Sinks are then identified based on the R-step transitive closure. A node is selected as a sink if it is not present in any other cluster. Finally, nodes, which do not violate the delay, relay load and sink capacity constraints, are assigned to the selected sink. The process is repeated until all nodes are assigned to the sinks. The incremental clustering algorithm is compared to the weighted recursive [14], the iterative greedy dominating set [17] and the augmenting placement. The results show that the incremental clustering's number of sinks is similar to that of the weighted recursive and less than that of the iterative greedy dominating set and the augmenting placement.

Find the Best Sink Location (BSL) and Minimise the Number of Sinks for a Predefined Minimum Operation Period (MSPOP)

Oyman and Ersoy [85] study two problems: *Find the Best Sink Location* (BSL) and *Minimise the Number of Sinks for a Predefined Minimum Operation Period* (MSPOP). In the BSL problem, the number of sinks is known prior to the deployment. The k-means clustering algorithm is used in this case to find the positions of the sinks, i.e. in the centres of disjoint clusters. In the MSPOP problem, the minimum required network lifetime is given. The algorithm deploys sinks one by one while evaluating the network lifetime. The search stops when the desired network lifetime is reached. Similar to the BSL problem, the sinks' positions are identified using the k-means clustering algorithm.

2.7.2 Fixed Number of Sinks

Cluster-Based Sampling (CBS)

Cambazard *et al.* [28] propose the *Cluster-based Sampling* (CBS) algorithm to find the optimal positions to deploy a given number of sinks to minimise the total Euclidean distance from all nodes to the two nearest sinks. CBS uses the concept of the k-means clustering algorithm. In the k-means clustering algorithm, a network is divided into disjoint clusters and each sink is placed in the centre of a cluster. CBS applies a variant of the k-means clustering algorithm to compute overlapping clusters. It starts by selecting the positions of k sinks randomly from the available nodes. Then, the overlapping clusters are formed, where each node joins two clusters, i.e. the clusters of its nearest and second nearest sinks. The centroid of each cluster is calculated and each sink is moved to the node's location near the new centroid. The iteration stops when no further improvements can be made in terms of the total distance of all nodes to two nearest sinks. CBS results are compared to the optimal solution found using mixed integer linear programming, where in all scenarios, CBS's total distances are not more than 0.05% longer.

Cluster Balancing

Mahmud *et al.* [73] present heuristic algorithms for partitioning the networks into k disjoint clusters and place one sink for each cluster to minimise and balance the total energy consumption of all clusters. They find the optimal location of the sink such that the total shortest hop distance of the cluster is minimised. It is basically constructing the connectivity graph and computing the total shortest hop distance by trying to deploy a sink in each possible location between two nearby nodes. The heuristic works in two phases. In the first phase, it creates k initial clusters by using a greedy approach. Initially, there are n clusters with one node in each cluster. Then, the algorithm repeatedly finds a cluster with the smallest total shortest hop distance and merges it with the best neighbouring cluster that minimises the total shortest hop distance of the resulting merged cluster. In the second phase, the algorithm keeps moving one node from the largest total shortest hop distance cluster to a neighbouring cluster with the smallest total shortest hop distance until the balance is reached.

Maximally Balanced Connected Partition (MBCP)

Slama *et al.* [103] extend the problem of *Maximally Balanced Connected Partition* (MBCP) to deploy multiple sinks. The MBCP algorithm partitions a network into connected sub-networks of equal size. It firstly divides the network into two connected sub-networks with equal number of nodes. To have more sub-networks, MBCP divides the sub-networks again. In each sub-network, a sink is randomly deployed. The simulation results show that this technique can prolong the network lifetime around 10%–20% longer compared to the random deployment.

Iterative Decomposition (DECOMP)

Ning and Cassandras [82] address the problem of optimally determining the location of sinks to minimise communication power of nodes which are directly connected

to the sinks. For reliability, each node is required to connect to at least p sinks but each sink can only accept at most q connections. The problem is formulated as a Mixed Integer Non-linear Programming (MINLP) problem. It tries to minimise the total transmission power of the nodes. Since the MINLP solver has the scalability issue and the optimal solution depends on the initial feasible solution, the authors propose the *Iterative Decomposition* (DECOMP) algorithm and use random placement for initial locations. The iteration consists of two steps: optimise the node-sink connections by finding the minimum power of each node to connect to p sinks and optimise each sink's location by moving it to the centre of mass of all nodes connecting to it. This process stops when no further improvements on the sinks' locations can be made. The simulation results show that DECOMP finds the same best solution with 10%–60% shorter runtime compared to the MINLP solver for small network cases, i.e. up to 75 nodes. MINLP is not scalable and thus it cannot find solutions for large cases.

Global and 1hop

Vincze *et al.* [115] present two algorithms for multiple sink deployment, namely *Global* and *1hop*. Global is a centralised iterative algorithm, where in each step the sinks form clusters by grouping nodes which are closer to them. Then, they find their new positions by finding the centroid of the clusters, i.e. locations where the sinks' resultant vector is zero. 1hop is the distributed version of the algorithm, where the sinks only know location information of neighbouring nodes. After collecting messages for t time period, the sinks can approximate the locations of distant nodes using the number of nodes that actually send packets through the sinks' neighbouring nodes. Based on this assumption, the authors extend the two algorithms to *Global Relocation* and *1hop Relocation*. Both algorithms relocate the sinks from time to time during the network operation to extend the lifetime. In the simulation, 1hop Relocation can prolong around 30% of the network lifetime compared to Global Relocation, and about 50% compared to Global and the random deployment.

Coverage and Latency Aware Actor Placement (COLA)

Akkaya and Younis [8] propose *Coverage and Latency Aware Actor Placement* (COLA), a heuristic algorithm that considers both delay requirement and coverage by deploying a number of sinks. COLA initially distributes the sinks evenly in the region for maximising coverage. The region is divided into equal sized cells as many as the number of sinks. Each sink is then placed at the centre of one cell. After that, all nodes select their nearest sinks to form clusters. Each sink then reposition itself at a location that enables minimum latency in data collection. COLA uses vertex 1-centre formulation to pick new locations for sinks. Firstly, it creates the minimum distance matrix of all nodes within a cluster. Then for each node, it finds the longest path from that node to any other nodes by searching in the corresponding row of the matrix. Finally, the smallest value is picked among the list of the longest path. The vertex 1-centre will be the location of the node which has this smallest value in its row. Simulation results show that COLA has about 30% increase in coverage and up to 40% delay reduction compared to the random deployment.

Genetic Algorithm for Hop Count Optimisation (GAHO) and Genetic Algorithm for Distance Optimisation (GADO)

Youssef and Younis [124] propose two algorithms, namely *Genetic Algorithm for Hop Count Optimisation* (GAHO) and *Genetic Algorithm for Distance Optimisation* (GADO), using genetic algorithms. The two algorithms deploy a given number of sinks to reduce packet latency. GAHO tries to minimise the number of hops between a node and one of the sinks, while GADO uses distances as the cost factor for optimisation instead of hop counts. The sink placement problem is viewed as a cluster assignment problem where a sink is placed at or close to the geometric centroid of all nodes in the same cluster. For a small number of sinks, GADO's topologies achieve the lowest delivery latency compared to topologies of GAHO,

COLA [8] and random deployment, which are higher by 12.5%, 75% and 125%, respectively. The simulations also indicate that the latency decreases as the number of sinks increases and for a large number of sinks, the algorithms perform very close to each other.

P-Median Problem (PMP)

Luo *et al.* [72] present a heuristic algorithm to deploy a given number of sinks to improve energy efficiency by shortening the distance between nodes and sinks. The algorithm deploys multiple sinks and optimises the total minimum weighted distance by taking into account the demand generated at nodes. Firstly, it will find the best position for one sink by moving it across all candidate positions. The initial deployment set is constructed by placing all sinks at the best identified position. Then, the algorithm tries to find the best positions for them one by one by keeping the other sinks untouched. The heuristic algorithm is proved to be applicable to large scale WSNs as the simulation time increases linearly with the increment of the number of sinks.

Greedy and Local Search Algorithms for Sink Positioning

Bagdanov *et al.* [22] study the problem of sink positioning for maximising the data rate while minimising the energy consumption of the network. By assuming that the sinks are located only at node positions, they propose two heuristics: a greedy algorithm and a local search algorithm. The greedy algorithm deploys sinks incrementally. That is selecting the sinks one by one to improve the data rate as much as possible, while keeping the positions of the deployed sinks fixed. The local search algorithm, on the other hand, starts with a random placement of sinks. Then, it tries to relocate any of the sinks to improve the data rate. After several iterations with random initial configuration, the deployment with the highest data rate is

picked as the best solution. The simulation results show that the local search algorithm produces a better sink positioning that yields a better power consumption than the greedy algorithm. The power consumption of both algorithms is similar for up to five sinks, but for a larger number of sinks, i.e. up to 20, the local search algorithm only consumes about half as much power as the greedy one.

Multi-hop Traffic-flow Weight Placement (MTWP)

Zhou *et al.* [128] propose the *Multi-hop Traffic-flow Weight Placement* (MTWP) algorithm to choose some nodes as sinks to maximise the throughput of the network. MTWP is an iterative algorithm to determine the best location of a given number of sinks. In each iteration, the node with the highest traffic-flow weight will be selected as a sink. MTWP is computed by taking into account the number of nodes and sinks, traffic demand, locations of sinks and interference from existing sinks. In the simulation, MTWP has been shown to achieve around 10%–30% more throughput than the random deployment.

2.7.3 Discussion

Table 6 shows the comparisons of the existing algorithms for the multiple sink placement problem. We compare the algorithms based on their objectives, the deployment locations and the constraints assumed. In the constrained deployment, sinks can only be placed at candidate locations. If the deployment locations are unconstrained, sinks can be placed anywhere. Co-located deployment means nodes and sinks' locations are overlapping. In this literature study, most of the algorithms use a linear programming method for small cases, where a network only consists of several nodes, and a heuristic or local search algorithm for bigger cases.

To be robust to sink failure, it is necessary for each node to be able to communicate with more than one sink. Among all existing algorithms, only CBS [28], 2-Connectivity [58] and DECOMP [82] address the fault-tolerant issue in multiple

Table 6: Comparison of existing multiple sink placement algorithms

Algorithms	Objectives	Deployment Locations	Additional Constraints
Minimise the Number of Sinks			
HOMP [122]	↓, balance energy	constrained	hop count
2-Connectivity [58]	fault-tolerance (2-connected)	unconstrained	–
Greedy Placement [91]	↑ bandwidth,	co-located	link, node, sink capacity
Negative Selection [120]	↓ delay, ↓ hop count	co-located	–
OPEN/CLOSE [89]	↓ cost, load balancing	constrained	node capacity
Iter. Greedy DS [17]	QoS (delay, load, capacity)	co-located	hop count, load, capacity
Weighted Recursive [14]	QoS (delay, load, capacity)	co-located	hop count, load, capacity
Incr. Clustering [111]	QoS (delay, load, capacity)	co-located	hop count, load, capacity
MSPOP [85]	required lifetime	unconstrained	minimum required lifetime
Fixed Number of Sinks			
BSL [85]	efficient clustering	unconstrained	–
CBS [28]	↓ distance, fault-tolerance	co-located	–
Cluster Balancing [73]	↓, balance energy	unconstrained	–
MBCP [103]	↑ lifetime	unconstrained	–
DECOMP [82]	↓ energy, fault-tolerance	unconstrained	–
Global, 1hop [115]	↓ energy	unconstrained	–
COLA [8]	↓ delay, ↑ coverage	co-located	–
GAHO, GADO [124]	↓ delay	unconstrained	–
PMP [72]	↑ lifetime	constrained	node demand
Greedy, Local Search [22]	↑ data rate, ↓ energy	co-located	–
MTWP [128]	↑ throughput	co-located	–

sink placement. With an assumption that the communication to sinks is within one hop, CBS and DECOMP require each node to be able to communicate to two sinks for fault-tolerance. Unlike CBS and DECOMP, 2-Connectivity assumes multi-hop communication among sinks and deploys sinks to make the network of sinks become 2-connected. Research opportunities arise as these three algorithms do not consider hop count limit in the algorithm designs. Therefore, there is a gap in the research literature for a multiple sink placement algorithm to design multi-hop networks where each node can communicate to multiple sinks with constrained path length. In addition, since the problems of deploying sinks and relays are solved separately in the literature, the second research opportunity is to minimise the total combined cost of sink and relay deployment by taking into account the fault-tolerant requirements. Our designed algorithms in Chapter 7 address these problems.

Algorithm 1: GRASP

Input : *max_iterations*, *seed*
Output: *best_solution*

```
1: Read-Input()
2: for k ← 1 to max_iterations do
3:     solution ← Greedy-Randomised-Construction(seed)
4:     solution ← Local-Search(solution)
5:     Update-Solution(solution, best_solution)
6: end for
7: return best_solution
```

2.8 Greedy Randomized Adaptive Search Procedures (GRASP)

GRASP [41, 42, 96] is a metaheuristic which captures good features of pure greedy algorithms and random construction procedures. It is an iterative process. In each iteration, it consists of two phases: the construction phase and the local search phase. The construction phase builds a feasible solution as a good starting solution for the local search phase. At each construction iteration, the next element to be chosen is determined by ordering all elements in a candidate list with respect to a greedy function that estimates the benefit of selecting each element. The probabilistic component of a GRASP is characterised by randomly choosing one of the best possible candidates in the list, instead of the overall best one. Since the solution produced by the construction phase is not necessarily the local optimum, the local search phase works iteratively to replace the current solution with a better one from its neighbourhood. It terminates when there are no better solutions available. The generic GRASP implementation for minimisation is given in Algorithm 1, in which *max_iterations* iterations are performed and *seed* is used as the initial seed for the pseudorandom number generator. When we increase the number of iterations, the computation time increases, but we will get a better solution.

Algorithm 2 gives the pseudocode for the construction phase. At each iteration, instead of selecting the best element, a Restricted Candidate List (RCL) is built by greedily selecting the best elements that can be incorporated to the current partial

Algorithm 2: Greedy-Randomised-Construction

Input : *seed*
Output: *solution*
1: *solution* $\leftarrow \emptyset$
2: Evaluate the incremental costs of the candidate elements
3: **while** *solution* is not a complete solution **do**
4: Build the restricted candidate list (RCL)
5: Select an element s from the RCL at random
6: *solution* $\leftarrow$ *solution* $\cup \{s\}$
7: Reevaluate the incremental costs
8: **end while**
9: **return** *solution*

Algorithm 3: Local-Search

Input : *solution*
Output: *solution*
1: **while** *solution* is not locally optimal **do**
2: Find $s' \in N(solution)$ with $f(s') < f(solution)$
3: *solution* $\leftarrow s'$
4: **end while**
5: **return** *solution*

solution and have the smallest incremental costs. An element e, which is associated with a cost $c(e)$, is included in the RCL if $c(e) \in [c_{\min}, c_{\min} + \alpha(c_{\max} - c_{\min})]$, where $0 \leq \alpha \leq 1$, while $c_{\min}$ and $c_{\max}$ denote, respectively, the smallest and the largest incremental costs. The case $\alpha = 0$ corresponds to a pure greedy algorithm, while $\alpha = 1$ is equivalent to a random construction. Each element is then randomly selected from those in the RCL. After the selected element is incorporated to the partial solution, RCL is updated and the incremental costs are reevaluated.

The solution that resulted from the construction phase is not necessarily the local optimum, so local search is utilised to improve it. A local search algorithm works in an iterative fashion by replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found. The pseudocode of a basic local search algorithm is given in Algorithm 3.

GRASP has been shown to be very powerful in solving combinatorial problems as its results are close to the optimal. Some applications of GRASP including the set covering problem [41], the Steiner tree problem [77, 78], power system transmission

network planning [21] and a capacitated location problem [35]. We refer to Resende and Ribeiro [96] for some references focusing the main applications of GRASP. In this thesis, we utilise the GRASP technique for the relay placement problem in Chapter 5 and 6, as well as the relay and sink placement problem in Chapter 7.

2.9 Summary

We identify the following gaps in the research literature for a MAC protocol and topology planning algorithms. Based on the existing MAC protocols that we reviewed in this thesis, none of them address all of the requirements for emergency response WSNs, i.e. behaves energy-efficiently when the traffic load is light, achieves high delivery rate and low latency when the traffic load increases, adapts to traffic fluctuations and topology changes, supports packet prioritisation and has fair packet deliveries in both normal and emergency situations. Our novel MAC protocol in Chapter 4 is designed to satisfy all of these criteria.

Based on the state-of-the-art relay placement algorithms that we reviewed in this thesis, we identify two research opportunities. Firstly, none of the algorithms address the single-tiered, constrained partial fault-tolerant relay placement problem for $k \geq 2$. Secondly, none of them take into account a path length constraint. Our solution that solves these problems is presented in Chapter 5. Another solution that relaxes the k-disjointness using centrality calculation is presented in Chapter 6. Based on the reviewed sink placement algorithms, none of them constrained path length and consider minimising the total combined cost of sink and relay deployment for fault-tolerance. Our solutions in Chapter 7 address these problems. We use the GRASP technique to design these topology planning algorithms.

Chapter 3

Research Methodology

3.1 Introduction

In this chapter, we summarise the general assumptions we made for the Wireless Sensor Networks (WSNs), list the requirements for the MAC protocol design and deployment planning, describe the simulation tool for the implementation of our algorithms, and define performance metrics for evaluating the performance of our algorithms against existing algorithms in the literature. All results presented in this thesis are based solely on simulation study. We choose to implement our approaches using simulations because of the practicalities of design evaluations, especially when we conduct experiments in various environments using different system configurations.

3.2 WSN Model and General Assumptions

In this thesis, we use some assumptions in our simulations for purpose of simplicity. The assumptions and the implication to the real-world situation are discussed below. We note that the assumptions made here are not only for emergency response as it is only one of the extreme applications.

1. *Perturbed grid deployment.* The initial WSN is connected, which means there is a routing path between every sensor node in the network. We make no assumptions on the geographical or physical properties of the area in which the WSN is to be deployed. However, for the ease of simulation purposes, we deploy up to 100 nodes within randomly perturbed grids of a two-dimensional network area. In the perturbed grid deployment, a sensor node is placed inside one unit grid square and the coordinate locations within each grid square are randomly perturbed. This is an approximation of manual deployment of sensor nodes, such as in a building or a city that has regular symmetry. The shape of the sensor field does not influence the performance of the MAC protocol and topology planning algorithms.

2. *Homogenous network.* All nodes are homogenous, i.e. they have identical hardware specification and capabilities, equal amount of initial energy, which is not time-dependent, and also the same level of power consumption for the same task, e.g. send or receive messages. Even though in reality all nodes are different, for example they have different initial energy so they may die in different order than what is expected, this assumption will not affect our simulation results. Our solutions allow the network to continue to receive data from all other nodes, and so we have time to replace a depleted battery without losing data.

3. *Static network.* All nodes are static, so they remain in the same position during the simulation period. This assumption is realistic because sensor nodes have no mobility, except if they are attached to moving agents, such as humans, robots, or vehicles. Moreover, the focus of this research is on in-building nodes, and not on-person nodes which would necessitate mobility.

4. *Single radio and omni-directional antenna.* Each node is equipped with a single radio transceiver and an omni-directional antenna. In a single radio architecture, nodes cannot transmit and receive data simultaneously. For

simplicity we also assume that all nodes operate in a single channel, even though the multi-channel techniques could be used to complement our proposed techniques to yield even better reliability.

5. *Bi-directional links.* The radio links are bi-directional. So, two nodes are neighbours if and only if they are within transmission range of each other. We use this assumption to simplify our simulations due to the fact that there is usually a site survey before sensor node deployment to know which nodes a sensor node can communicate with, and we assume that locations have been chosen to ensure bi-directional links. In reality, a link can be asymmetric and it affects the connectivity graph, where a node v has node w as its neighbour but w does not have v as its neighbour. Link asymmetry is beyond the scope of this thesis.

6. *Unit disk graph model with fixed transmission range.* To have bi-directional links in our simulations, we assume that the communication graph follows the unit disk graph model. In this model, a node v can communicate directly to a node w if the distance between these two nodes $\leq r$, where r is the transmission range of the nodes. This model represents the transmission range as an ideal circle. In the simulation, we use 10 metres as the transmission range of a node for communicating with its neighbours. This assumption is realistic for 0 dBm transmission power in an indoor environments [119].

7. *Single sink and multiple sinks.* The WSN can have either one static sink that gather data from all sensor nodes, or multiple sinks.

3.3 WSN Requirements

WSNs must be designed with fault-tolerance in mind in order to be resilient to network dynamics, including traffic fluctuations and topology changes. The extreme case to these dynamics would be during emergency response, such as in fire, flood

and volcano monitoring. In the following, we list the network requirements for such an application:

1. *Traffic load* is light during normal day-to-day monitoring but increases when an emergency event occurs. In this thesis, we focus on in-building traffic type, where during non-emergency, sensor nodes report their sensed data not very often, for example one message every 30 seconds or one minute. However, in emergency monitoring, they generate data in a higher rate, approximately every 10 seconds.

2. The network must achieve *high delivery ratio* in both normal and emergency monitoring from all sensor nodes in the network. In order for a MAC protocol to achieve high delivery ratio, especially when topology changes, the physical topology must ensure the availability of k *disjoint paths* that can tolerate $k-1$ node failures.

3. Normal monitoring is usually delay-tolerant, but emergency monitoring is not. Therefore, the network must offer *low delivery latency* when a hazard occurs. For example, a high priority message must reach a sink within 10 seconds. Delivery latency of a MAC protocol is influenced by *the length of the paths* from sensor nodes to reach the sink, so we bound the path length in our topology planning algorithms.

4. Sensor nodes must behave in *energy-efficient* manner to prolong its lifetime. Only those who participate in emergency monitoring can sacrifice their energy efficiency to achieve high delivery ratio and low latency. By not switching all nodes into emergency monitoring mode, their battery will not be drain when a false alarm happens.

5. Even though the network is designed to be fault-tolerant, its *deployment cost* must also be minimised. This is due to the fact that installing many devices comes at a cost that includes not only the hardware purchase but also the installation and ongoing maintenance.

3.4 Simulation Model

While our topology planning algorithms are implemented in C++, the communication protocols are implemented in the open-source network simulator ns-2 version 2.33 [2]. Ns-2 is a discrete-event, packet-level network simulator that is widely used for WSN and other network simulations, such as wired, wireless and satellite networks. In its distribution, it has a large number of libraries and tools for WSN simulations. Ns-2's main functionality and detailed protocol implementation are written in C++, while the simulation configuration is controlled by Tcl scripts. We will describe the simulation components that we used in the following subsections.

3.4.1 Input and Output

Ns-2 takes as input Tcl scripts and topology files. In the Tcl scripts, we define simulation configurations, including global configurations for nodes (protocol stack, radio propagation model, antenna type, energy model, etc that will be explained in details in the consecutive subsections), a simulation event scheduler to indicate when nodes should start or stop transmitting packets, traffic load, network topology and output files.

To generate static network topologies in a perturbed grid deployment, we implement a topology generator using C++. The two-dimensional network area is divided into grid cells, where one node is placed inside one unit grid square and the coordinates are randomly perturbed. The topology generator produces topology files using ns-2 topology format, where the coordinates of each node in the network are written as the following:

```
$node_(<node_id>) set X_ <x_coordinate>
$node_(<node_id>) set Y_ <y_coordinate>
$node_(<node_id>) set Z_ <z_coordinate>
```

All variables are self-explanatory. The topology files are read as input by ns-2 and used to construct the topology to be simulated.

Ns-2 produces trace files as its output to allow packet tracing, where we can get information such as node that sends or receives a packet, time when the action happened, layer where it happened, the remaining energy of the node, etc. We can also generate our personalised trace files to save CPU resources by selecting which parameters to be traced.

3.4.2 Protocol Stack

The five core layers of a sensor node consist of the application layer, transport layer, routing layer, MAC layer and physical layer. At a simulated node, the application layer generates data packets at regular time intervals and passes them down to the transport layer. At the transport layer, a node implements User Datagram Protocol (UDP), where no connection setup is needed prior to data transfer. It adds nothing to the packets and passes them to the routing layer [57]. Our proposed technique, ER-MAC in Chapter 4, has its own forward-to-parent routing mechanism, where every node sends packets to their parent nodes in the routing tree. Therefore, upon reception of a packet, ER-MAC arranges channel access and passes the packet to the physical layer for transmission. Later in the evaluation of ER-MAC, we use Z-MAC [97] as a comparison. Z-MAC does not have any routing policies. So, we implement Shortest Path Tree Routing (STR), which is similar to the routing protocol from Collection Tree Protocol (CTP) [47], at the routing layer. While CTP uses expected transmissions as the cost metric, our implementation of STR uses hop counts. When STR receives a packet, it finds a one-hop neighbour that has the shortest route to the destination. After that, STR passes the packet down to Z-MAC for arranging channel access, and then Z-MAC passes it down to the physical layer for transmission. When a node receives a packet at another side of the transmission, the packet is passed up the stack until received by the application layer.

3.4.3 WirelessPhy Model

Each node has a WirelessPhy interface which represents the hardware interface and the properties of its radio. The interface puts transmission data, such as transmitted signal power, into the header of each packet. When a packet is received, the propagation model uses the transmission data to determine whether it has the minimum signal power to be received by the receiving node. If the received signal power is below the receiving threshold *RXThresh*, the packet will be dropped. This model approximates the Lucent WaveLAN Direct-Sequence Spread-Spectrum (DSSS) radio interface. We can specify the transmission range of a node by setting an appropriate value of the transmitted signal power and the receiving threshold. To get a 10-metre transmission range in ns-2, we set the transmitted signal power P_t equal to 5.35395e-05 and the receiving threshold *RXThresh* equal to 3.65262e-10. While *RXThresh* is set to ns-2 default value, we use a separate C program, i.e. `threshold.cc`, that comes with the ns-2 installation to compute the value of P_t.

The WirelessPhy interface also has a direct access to control the properties of a node's radio, which can be set to either on or off mode. When the radio is turned off, all received packets are discarded.

3.4.4 Radio Propagation Model

To represents the transmission range of a sensor node as an ideal circle, we select a simple wireless channel using the two-ray ground radio propagation model. The radio propagation model is used to predict the received signal power of each packet. Using a standard model, the received signal power at a distance d is calculated by

$$P_r(d) = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4 L} \tag{1}$$

$P_r(d)$ is the received signal power given a transmitter-receiver distance,
P_t is the transmitted signal power,
G_t is the antenna gain of the transmitter,

G_r is the antenna gain of the receiver,

h_t is the height of the transmit antenna above ground,

h_r is the height of the receive antenna above ground,

d is the distance between the transmitter and the receiver, and

L is the system loss in the transmit/receive circuitry.

The omni-directional antenna that we use in the simulation can radiate or receive energy equally well in all directions. It has the following specifications: $G_t = G_r = 1$, $h_t = h_r = 1.5$ metres, and no system loss ($L = 1$) [116].

3.4.5 Simulation Parameters

We set the transmission power of each sensor node at 0 dBm, where its transmission range in an indoor environment is about 10 metres [119]. Our simulation uses the energy model of the Tmote sky hardware [4]. Tmote sky, which is depicted in Figure 22, is a popular wireless sensor node. Its current consumption for radio transmitting at 0 dBm is 17.4 mA and for radio receiving is 19.7 mA. Tmote uses 2 $\times$ AA batteries. Each AA battery is 1.5 Volts and has energy up to 10,000 Joules. Note that we need one joule of energy to produce one watt of power for one second. Our simulation parameters are presented in Table 7. These parameters are used for our ns-2 simulations to evaluate the proposed ER-MAC protocol in Chapter 4 and the network performance of designed topologies in Chapter 8.

Figure 22: Tmote sky [4]

Table 7: Simulation parameters in ns-2

Simulation parameters	**Default value**
Transmission range	10 m
Transmit power (*txPower*)	17.4 mA × 3 V = 52.2 mW
Receive power (*rxPower*)	19.7 mA × 3 V = 59.1 mW
Idle power	19.7 mA × 3 V = 59.1 mW
Sleep power	1 μA × 3 V = 3 μW
Transition power	19.7 mA × 3 V = 59.1 mW
Transition time	580 μs
Node initial energy	20,000 J (2 × AA batteries)

3.5 Performance Metrics

In this thesis, communication protocols are implemented and evaluated in the network simulator ns-2. To measure the network performance in ns-2, we use one or more of the following metrics:

1. ***Average energy consumption per node*** is presented to compare the energy efficiency of communication protocols. Since our protocol design does not involve load balancing, we evaluate neither the lifetime of the network nor the maximum/minimum energy consumption of nodes. Without load balancing, the energy expenditure of nodes in the network is not balanced and thus these two metrics may be biased toward nodes that die earlier than expected. The average energy consumption per node is calculated as the total energy consumed by the entire network during the simulation period for listening, transmitting, receiving, switching from sleep to idle mode and vice versa, averaged over the total number of nodes in the network. The unit of energy is the Joule.

2. ***Packet delivery ratio*** is the total number of packets received at the sink divided by the total number of packets generated by the source nodes during the lifetime of an experiment. In our experiments, we distinguish between the delivery ratio of high and low priority packets, because we assume that there are two types of packets and the high priority ones must be delivered first.

So we expect that their delivery ratio is higher than the low priority ones. Delivery ratio takes a value in the interval [0, 1].

3. ***Average per packet latency*** measures the total time needed for each packet to reach the sink since it was sent by the source node, averaged over the total number of packets received at the sink. We also present the latency of high and low priority packets separately, because we expect to see that the latency of high priority packets is lower than the low priority ones. The unit of latency is the second.

4. ***Completeness of packets received*** is presented to show if the sink can have a balance of information from all sensor nodes in a network, no matter how far they are from the sink. Firstly, the network is divided into zones, where sensor nodes that have the same hop distance to the sink are grouped into one zone. Completeness measures the percentage of packets received per zone, where we distinguish the high priority packets from the low priority ones. We choose to present this metric rather than the fairness index [59] because we want to show the fairness over the packets' sources per zone, while the fairness index only shows the fairness of the whole network.

5. ***Connectivity*** is the percentage of alive source nodes that are still connected to the sink through multi-hop communication. A source node is counted as connected if at least one of its generated packets is received by the sink. We present this metric to show the adaptability of protocols when topology changes.

We implement and evaluate our topology planning algorithms in C++. We use C++ because we do not evaluate network protocols and operations when we evaluate the performance of the algorithms in planning a network. We will later evaluate the network performance of the designed topologies using ns-2. We use the following metrics to evaluate the performance of topology planning algorithms:

1. ***Number of table lookups*** measures the efficiency of an algorithm by the total number of table lookups. We count how many times an algorithm gets information from and updates information in the table representing the graph.

2. ***Storage capacity*** measures the efficiency of an algorithm in terms of the memory size required. This metric counts the total number of array cells needed to store graph's information. The unit of storage capacity is the cell.

3. ***Number of disjoint paths*** is the total number of disjoint paths per node found averaged over the total number of nodes in the network. We present this metric to compare the accuracy of algorithms in finding disjoint paths.

4. ***Percentage of nodes with disjoint paths*** shows the percentage of sensor nodes that have k disjoint paths over the total number of sensor nodes in the network. When we deploy fewer relay nodes in the network to reduce the deployment cost, some sensor nodes do not have k disjoint paths. Therefore, we present this metric to show the relationship between deploying fewer relays and the number of nodes that have k disjoint paths in the network.

5. ***Number of additional relay nodes*** measures the effectiveness of a relay deployment algorithm by the total number of relay nodes that are required to be deployed in the network. A relay deployment algorithm is the most effective if it places the fewest relays.

6. ***Number of sinks needed*** measures the effectiveness of a sink deployment algorithm by the total number of sinks deployed in the network. A sink deployment algorithm is the most effective if it places the fewest sinks to satisfy the deployment requirements.

7. ***Total sink cost*** also measures the effectiveness of a sink deployment algorithm but using the total cost of deployed sinks. Since the deployment costs of an individual sink may be different, we present this metric to show that the most effective sink deployment algorithm not only deploys the fewest sinks

but also finds the lowest cost solution. In the results presented, cost is shown in units.

8. ***Total sink and relay cost*** presents the total deployment cost of a sink and relay deployment algorithm, which includes the cost of sinks and the cost of relays. This metric is shown in units.

9. ***Number of devices*** shows the total number of deployed sinks and relays. This metric is presented because we cannot infer how many sinks and relays are deployed from the total cost metric. It is important to see the relationship between the increment of sink costs and the number of deployed sinks and relays. When the sink cost increases, we expect to see fewer deployed sinks and more relays, because some sinks are traded for relays to reduce the deployment cost.

10. ***Runtime*** is the total time needed for an algorithm to finish its execution. Even though topology planning is an offline process, shorter runtime is important because a topology planning algorithm may be executed several times to evaluate different topologies as a result of either moving or adding/removing a node. Runtime is measured in seconds.

All experiments are carried out in 2.40 GHz Intel Core2 Duo CPU with 4 GB of RAM. We present the simulation results as the mean values of multiple simulation runs, enough to achieve a 95% confidence in the standard error interval, which are shown as error bars in the results. In this thesis, we do not show error bars in line graphs and graphs with logarithmic scale to improve readability of the graphs.

Chapter 4

A Hybrid MAC Protocol for Emergency Response

4.1 Introduction

In this chapter, we present ER-MAC, a novel hybrid MAC protocol for emergency response Wireless Sensor Networks (WSNs). This protocol is energy-efficient during normal monitoring, achieves high delivery ratio and low latency for emergency monitoring, adapts to traffic and topology changes, prioritises high priority packets and supports fairness over the packets' sources. All of these design criteria make ER-MAC a novel contribution to the research literature.

For energy efficiency, ER-MAC adopts a TDMA approach to schedule collision-free transmission toward the sink. Sensor nodes wake up to transmit and receive messages according to their schedules, but otherwise switch into sleep mode to save energy. When an emergency event occurs, nodes involved in the emergency monitoring change their MAC protocol autonomously to *emergency mode* to allow contention in TDMA slots to cope with large volumes of traffic. This scheme trades off energy efficiency for higher delivery ratio and lower latency. Nodes involved in the emergency monitoring are nodes caught in hazard, their one-hop neighbours,

their ancestors (toward the sink) that receive emergency packets and the direct neighbours of the ancestors, while the rest of the network operates using the *normal mode* of ER-MAC.

A node in emergency mode may have neighbours which are not in emergency mode. Therefore, before contending for its neighbours' transmit slot, it has to listen for any activities on the channel. If it does not sense any activities, it may contend for the slot. Otherwise, it knows that the neighbour which is not in emergency mode is still using its transmit slot. In addition, to prevent a node sending an emergency packet while its receiver is sleeping, the first emergency packet is always sent in a scheduled transmission slot. This will allow the ancestors of the node toward the sink to switch their MAC protocol to the emergency mode when they receive the emergency packet. The implication of this method is the delivery latency of the first emergency packet is the same as the normal situation. However, ER-MAC guarantees fast deliveries of high priority packets when nodes that involved in the emergency monitoring have switched their MAC protocol to the emergency mode. This delay to switch is bounded by one data gathering cycle period.

ER-MAC maintains two priority queues to separate high priority packets from low priority ones. The low priority packets are sent if the high priority queue is empty. Therefore, the high priority packets are propagated faster to the sink. ER-MAC also offers a synchronised and loose slot structure, where nodes can modify their schedules locally. This mechanism enables the addition of new nodes and removal of dead nodes without restarting the whole network.

The rest of this chapter is organised as follows. We formulate the problem definition in Section 4.2. We present the proposed ER-MAC protocol in Section 4.3. We show our simulation results in Section 4.4. Simulation results validate the performance of ER-MAC, which outperforms Z-MAC [97], a state-of-the-art hybrid MAC protocol, with higher delivery ratio and lower latency at low energy consumption.

4.2 Problem Definition

In this section, we describe some assumptions for the network and identify the requirements for our MAC protocol.

4.2.1 Assumptions

We assume a pre-deployed WSN for fire emergency that has a connected finite set of sensor nodes and one or more sinks, which are static. We also assume that there are two types of packets: high priority packets and low priority packets. The priority of a packet is determined based on its content. For example, data from temperature sensors can be tagged as high priority, while light measurements are considered as low priority. We assume these priority do not change when the WSN detects a hazard, but their reporting frequency increases, because high priority packets are more important than the low priority ones. Therefore, they must be delivered first either in normal or emergency monitoring.

In "fire emergency situation", a combination of sensors, such as smoke, temperature and CO [31], ION and CO [49], can collaborate to detect the presence of fire when its sensor reading is above a specified threshold. In this hazard situation, the WSN must be able to assist fire fighters by dynamically providing important information such as the location of the fire, the estimation of the spread of the fire, as well as evacuation routes [109] to both evacuees and the fire fighters.

We assume two different network situations: no-fire and in-fire. No-fire is the normal situation where the communication is delay-tolerant and must be energy-efficient to prolong the network lifetime. When a sensor node or a group of sensor nodes senses fire, it changes the MAC behaviour to emergency mode autonomously. The communication of in-fire nodes is not delay-tolerant and energy efficiency is not as important as achieving high delivery ratio and low latency. However, the rest of the network that is not involved in the fire monitoring must be energy-efficient.

4.2.2 Requirements for MAC

When designing the MAC protocol for emergency response, there are several important factors that have to be taken into account:

1. ***Traffic load*** of the network depends on the reporting frequency of the sensor nodes. It is light during normal monitoring, but increases significantly when an emergency occurs and may be unbalanced. The MAC protocol is expected to offer reliable delivery when the traffic load increases. That is, when the WSN generates more traffic, it does not lose performance as specified by the metrics in Section 3.5.

2. ***Energy efficiency*** is one of the most critical factors for WSN applications. The lower the energy consumed by each node, the longer the WSN can perform its mission. Therefore, during normal day-to-day monitoring, the network must be energy-efficient to prolong its lifetime. However, energy efficiency can be sacrificed for low latency and high delivery ratio during emergency.

3. Successful communication of the WSN not only requires a robust and reliable communication protocol to transport the important messages to the sink, but also depends on ***delivery latency***. Normal monitoring is delay-tolerant, but emergency monitoring is not, as high priority packets need timely delivery at the sink.

4. The MAC protocol has to achieve high ***delivery ratio*** in both normal and emergency situations.

5. ***Detection delay*** must be bounded, so any messages, especially the emergency ones, can reach the sink within predictable duration.

According to these requirements, the MAC protocol must be energy-efficient when the network performs normal monitoring, has low packet latency and high packet delivery ratio when the network monitors a hazard, adapts to very heavy traffic and

topology changes, prioritises high priority packets and has fair packet deliveries. Since none of the existing MAC protocols reviewed in Section 2.3 are designed for emergency response, none of them address all of our MAC protocol requirements. Specifically, none of them try to address both packet prioritisation and fairness issues at the same time. Hence, we design ER-MAC that satisfies all of these design criteria. Packet prioritisation is necessary during emergency response to prioritise high priority packets, which are more important than the low priority ones. Fairness is important when a hazard occurs, so the sink can receive complete information from all sensor nodes in the network and monitor the spread of the hazard.

4.3 ER-MAC Protocol Design

The main functions of ER-MAC are to:

1. establish a data gathering tree with a sink as the root of the tree and retrieve neighbourhood connectivity (topology discovery),
2. establish nodes' schedules (TDMA slot assignment),
3. manage local time synchronisation to minimise clock drifts,
4. manage two priority queues for different priority packets,
5. respond to emergency events by changing MAC behaviour (MAC prioritisation) to cope with large volume of traffic, and
6. manage the network when the topology changes.

ER-MAC initially communicates using the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with a random backoff mechanism to avoid collision, where each transmission follows the sequence of RTS/CTS/DATA/ACK. During the startup phase, the data gathering tree and TDMA schedules for exclusive communication among nodes are created. We integrate routing functions into ER-MAC

because even though it is less flexible [88], it is known to be more efficient in a network protocol design for WSNs [79]. Firstly, it can improve energy efficiency by eliminating the use of unnecessary protocol overheads at both MAC and routing layers. Secondly, it can improve resource management by sharing resources between the two layers. In the data gathering tree, every node (except the sink) has one parent node and every non-leaf node (include the sink) has one or more children. The TDMA schedules enable each node to send its own data and forward its descendants' data to its parent in collision-free slots. Each node also has a special slot to broadcast a synchronisation message or any messages to its children. Besides contention-free slots, ER-MAC has a contention period at the end of each frame to support the addition of new nodes.

ER-MAC uses two queues for two kinds of packets: high and low priority packets. The low priority packets are transmitted only if the high priority queue is empty. Furthermore, inside a queue, packets are ordered based on their *slack*. That is, the time remaining until the packet deadline expires. The deadline is assigned by the WSN application to specify the desired bound on the end-to-end latency and is initialised by a source node. When a queue is full, the packet with the shortest slack is dropped because it may miss the deadline.

With the normal mode of ER-MAC, a node only wakes up to transmit and receive messages in its scheduled time slots, and spends most of its lifetime in sleep mode to conserve energy. However, when an emergency event occurs, nodes that are affected by the hazard change their MAC to emergency mode. In the emergency mode, ER-MAC allows nodes within one-hop neighbourhood to contend for a slot if they have priority data to be sent and if the schedule does not conflict with their two-hop neighbours' schedules. In the contention, the owner of the slot has higher priority to use its own slot than the non-owner of the slot, because it can transmit a packet immediately if it has a high priority packet to send. Furthermore, during an emergency, a node that has changed its MAC to emergency mode will wake up in the beginning of each TDMA slot for possible reception of packets.

4.3.1 Topology Discovery

During the initial startup phase, the sink initiates the tree construction using a simple flooding mechanism. Our process is similar to the hop tree configuration of the Periodic, Event-driven and Query-based (PEQ) routing protocol [24] and the level discovery phase of the Timing-sync Protocol for Sensor Networks (TPSN) [45]. However, in our context, the goal of the topology discovery is not only to setup a routing tree, but also to find neighbours and to track changes in the tree. Topology discovery is only performed once during the initial startup phase as nodes with ER-MAC can modify their schedules locally during the network lifetime.

The sink generates a *TOPOLOGY_DISCOVERY* message, which consists of:

1. *src_ID* is the sender of the message,
2. *hop_count* stores the number of hops to reach the sink,
3. *new_parent_id* stores the new parent ID of a node, and
4. *old_parent_id* stores a node's previous parent ID.

The format of a *TOPOLOGY_DISCOVERY* message is depicted in Figure 23. This message is broadcast by a node to find its prospective children, as well as a reply to its parent and a notification to its previous parent when it wants to change parent. A node replies its new parent, so the parent can add it to its children list. When choosing a new parent, which has shorter hop count to the sink, the node has to inform its previous parent to remove it from the parent's children list. Figure 24 illustrates a tree built for data gathering in a network of six nodes. A node has to record its parent ID because it will be used as the next hop destination in every packet transmission toward the sink. A node also needs to maintain a children list, so if it does not receive any messages from a particular child, it may know that the child is dead. We will discuss dead nodes later in Section 4.3.7.

The sink initialises *hop_count* as zero and leaves *new_parent_id* and *old_parent_id* as undefined. It broadcasts the message to its neighbours within its transmission range.

Field	*type*	*src_ID*	*hop_count*	*new_parent_id*	*old_parent_id*
Field size (bytes)	1	2	2	2	2

Figure 23: *TOPOLOGY_DISCOVERY* packet format

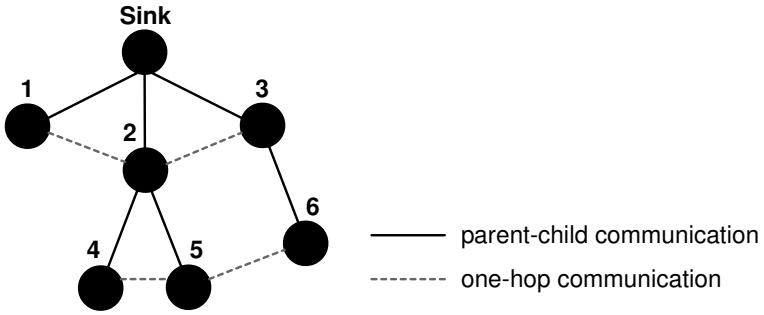


Figure 24: A data gathering tree of six nodes

In this phase, each node records the number of hop counts to the sink, its parent ID, a list of its children and its one-hop neighbour list. Communications among nodes during this phase use CSMA/CA with random-access to avoid collisions, because the TDMA schedules for exclusive communication have not been created yet.

Figure 25 shows the message exchange between a node, its parent, its child(ren) and a new parent during the topology discovery. Note that we do not show message overhearing in this figure because we want to focus the illustration on messages received and broadcast by a node. When a node receives its first *TOPOLOGY_DISCOVERY* message, it sets the sender of the message as its parent, increments the *hop_count* by one and sets it as its hop count to the sink. The node then stores its parent ID in *new_parent_id*, sets *old_parent_id* as undefined, waits for a random amount of time and re-broadcasts the message. If the node has already received a *TOPOLOGY_DISCOVERY* message before, it compares the new message's *hop_count* with its current hop count. If the new message's *hop_count* incremented by one is less than its hop count, it updates its parent ID and its hop count value. Then, it stores the new parent ID in the message's *new_parent_id*, the previous parent ID in *old_parent_id*, waits for a random amount of time and re-broadcasts the message.

Otherwise, if the new message's *hop_count* incremented by one is greater or equal to its hop count, the node ignores and does not re-broadcast the message.

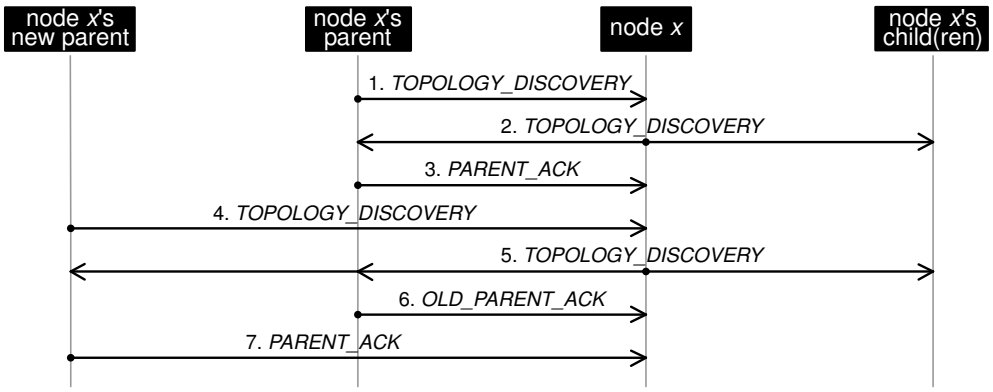


Figure 25: Message exchange in topology discovery

Upon receiving a *TOPOLOGY_DISCOVERY* message, a node also checks the message's *new_parent_id* and *old_parent_id*. If *new_parent_id* is the same as the node's ID, it adds the sender's ID to its list of children. If the node's ID is the same as *old_parent_id*, it removes the sender's ID from its list of children.

For reliability, a parent node replies its children with a *PARENT_ACK* message to confirm that each child has been added to its children list. If a node does not receive a *PARENT_ACK* message after broadcasting a *TOPOLOGY_DISCOVERY* message for a certain period of time (user parameter), it re-broadcasts the message. The node keeps broadcasting the *TOPOLOGY_DISCOVERY* message until it receives a *PARENT_ACK* message or exceeds the number of maximum retransmission. In another case, if a node updates its parent ID and its hop count value, it also needs a reply from its old parent after re-broadcasting the *TOPOLOGY_DISCOVERY* message. The old parent replies the node with an *OLD_PARENT_ACK* message to inform the node that it has been removed from the children list. If the node does not receive the *OLD_PARENT_ACK* message, it will re-broadcast the *TOPOLOGY_DISCOVERY* message. The *OLD_PARENT_ACK* message helps keeping the children list up to date. If the children list is not updated, the old parent may waste energy in idle listening, tries to receive some packets from the child for

several data gathering cycle before deciding to remove it from the list. The node will keep broadcasting the *TOPOLOGY_DISCOVERY* message until it receives the *PARENT_ACK* and the *OLD_PARENT_ACK* messages, or exceeds the number of maximum retransmission.

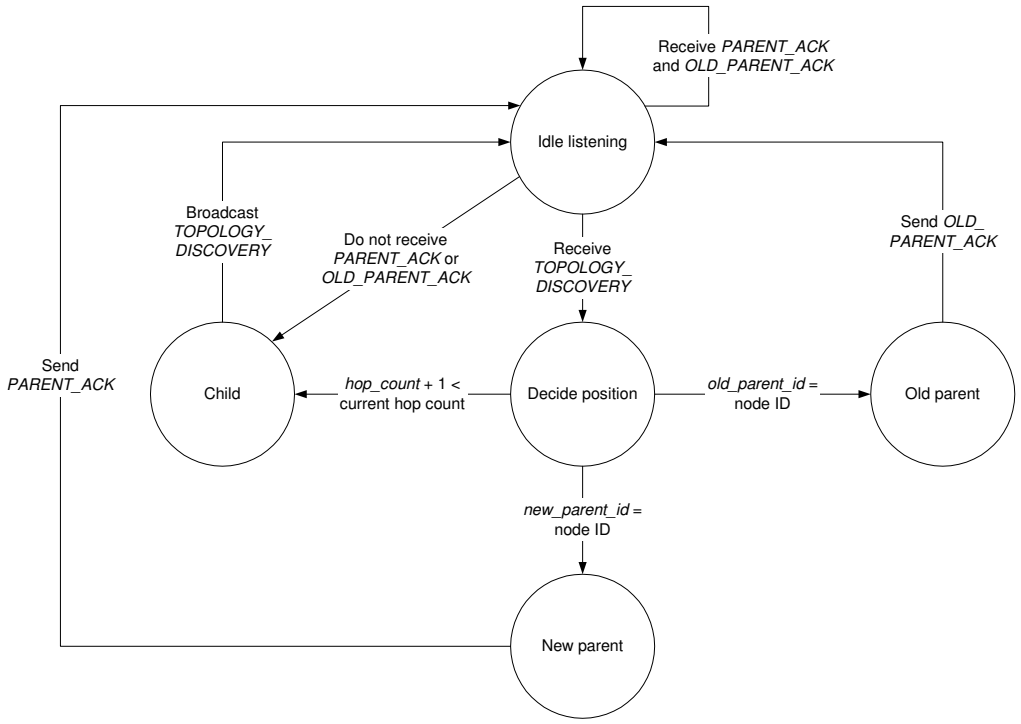


Figure 26: State transition diagram of topology discovery

During the topology discovery phase, a node may overhear transmissions from other nodes within its transmission range. The node records the senders of the messages as its one-hop neighbours in the one-hop neighbour list. This phase ends when all nodes in the network have already received the *TOPOLOGY_DISCOVERY* message. When this phase ends, each node knows the number of hops to reach the sink, its parent, the children list and the one-hop neighbour list. Figure 26 illustrates the state transition diagram of the topology discovery process.

4.3.2 TDMA Slot Assignment

During this phase, nodes perform slot assignment and exchange schedules, so no two nodes within a two-hop neighbourhood use the same slot. If two nodes are two hops away from each other and have the same time slot, their transmissions may collide at a node that is one hop away from both of them. At the end of this phase, each node maintains its own schedule, as well as its one-hop and two-hop neighbours' schedules to avoid schedule conflict. Our TDMA slot assignment follows a bottom-up approach, where a leaf node (a node with no children) starts the slot assignment. Our purpose of starting the slot assignment from the leaf nodes is to have transmission schedules that can support message flow toward the sink. During the TDMA slot assignment phase, all communications that are used to schedule conflict-free slots still use CSMA/CA.

Figure 27 shows the message exchange between a node, its parent and its two-hop neighbourhood during the TDMA slot assignment. A node deems itself as a leaf node if it has no children after broadcasting *TOPOLOGY_DISCOVERY* messages for a certain period of time. It selects its own time slot to send data to its parent. A leaf node always selects the smallest available slot. It then generates a *SCHEDULE_ANNOUNCEMENT* message, appends its schedule (the ID of the slot) and broadcasts the message to its one-hop neighbours. Nodes in its one-hop neighbourhood then re-broadcast this message to the two-hop neighbours.

When a node receives a *SCHEDULE_ANNOUNCEMENT* message, it copies the schedule into its one-hop neighbours' schedules if the sender of the message is its direct neighbour. Otherwise, the schedule is copied into the two-hop neighbours' schedules. All nodes that receive the *SCHEDULE_ANNOUNCEMENT* messages from the sender's one-hop neighbours know that they are two hops away from the sender. Every node within a two-hop neighbourhood of the message's sender checks if there is a possible conflict between its own schedule and the newly announced schedule. If it happens to be a conflict, the node generates a

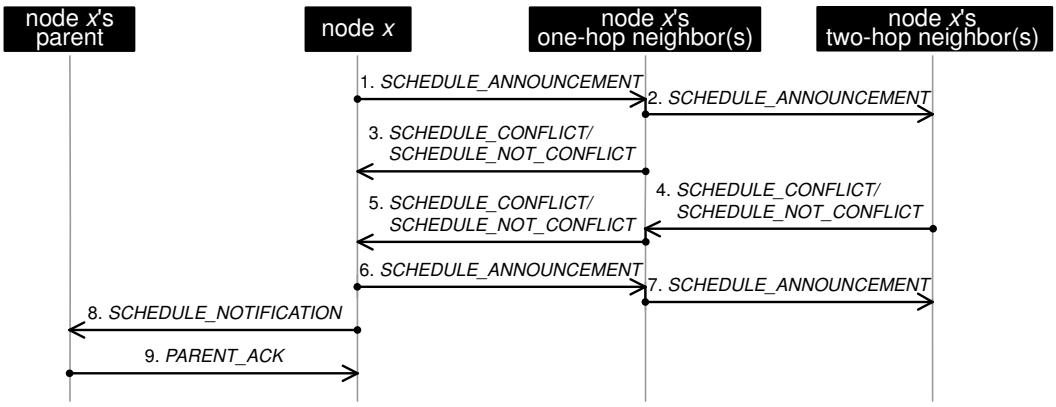


Figure 27: Message exchange in TDMA slot assignment

SCHEDULE_CONFLICT message, appends its schedule to the message and sends it back to the sender of the *SCHEDULE_ANNOUNCEMENT* message. When the sender of the *SCHEDULE_ANNOUNCEMENT* message receives the *SCHEDULE_CONFLICT*, it updates the conflict schedule in either its one-hop neighbours' schedules or its two-hop neighbours' schedules, depends on the origin of the *SCHEDULE_CONFLICT* message. Then, it re-assigns the schedule and broadcasts a new *SCHEDULE_ANNOUNCEMENT* to its two-hop neighbourhood.

Keeping in mind that collisions on the channel exist during this random-access period, we take into account lost and duplicate messages. Because the *SCHEDULE_CONFLICT* message may be lost during transmission, we make other neighbours that receive the *SCHEDULE_ANNOUNCEMENT* message send *SCHEDULE_NOT_CONFLICT* messages to the *SCHEDULE_ANNOUNCEMENT*'s sender if their schedules do not conflict. In order to reduce further collisions, the sender of the *SCHEDULE_ANNOUNCEMENT* saves a list of neighbours' ID whom it receives the *SCHEDULE_NOT_CONFLICT* messages from and appends this list to the *SCHEDULE_ANNOUNCEMENT* message. Neighbours do not send the *SCHEDULE_NOT_CONFLICT* messages if they are already in the list. The sender of the *SCHEDULE_ANNOUNCEMENT* is convinced that its schedule does not

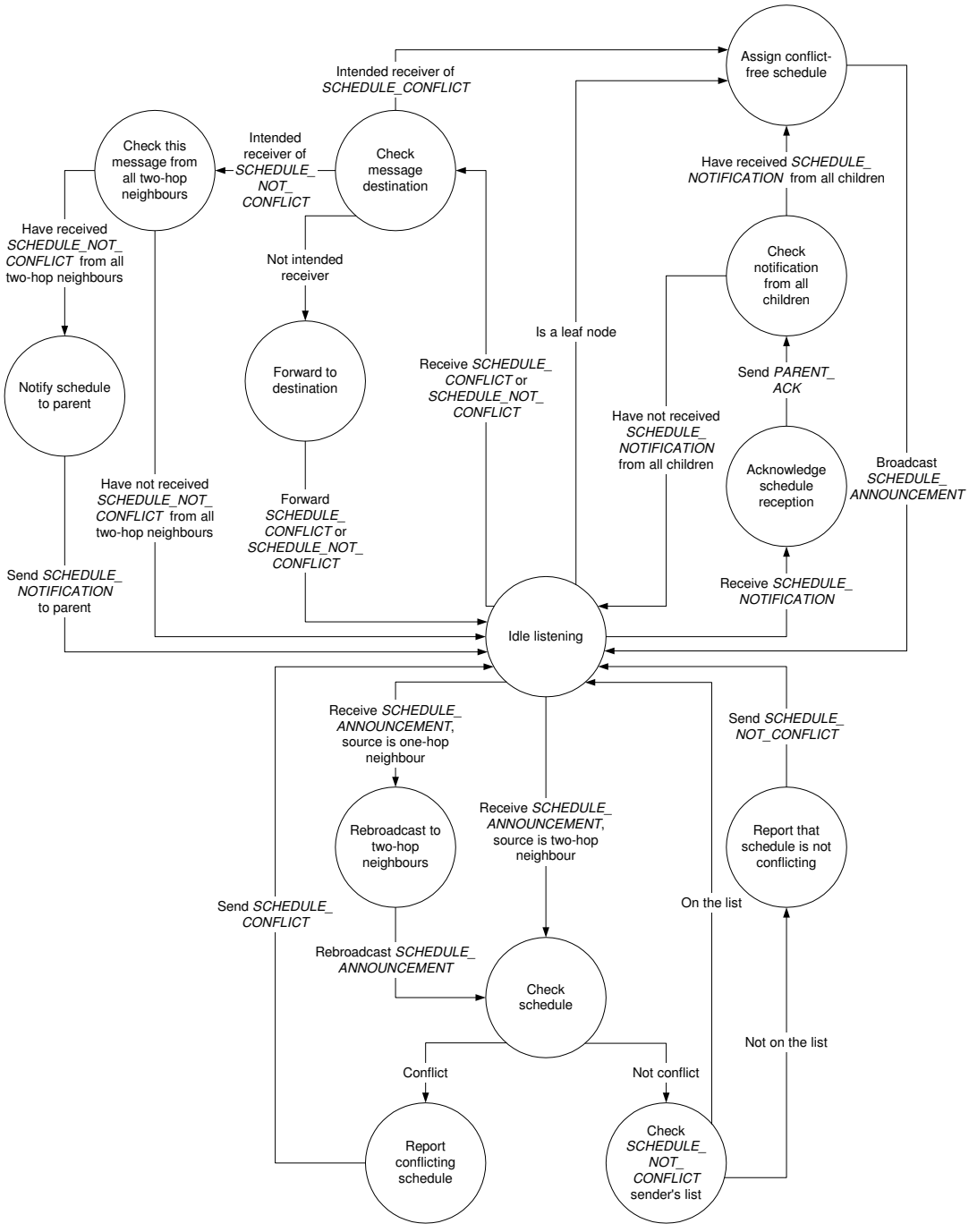


Figure 28: State transition diagram of TDMA slot assignment

conflict with its two-hop neighbours' schedules if it receives no more *SCHEDULE_NOT_CONFLICT* messages from its two-hop neighbourhood after broadcasting the *SCHEDULE_ANNOUNCEMENT* messages several times. The node then sends its assigned schedule in a *SCHEDULE_NOTIFICATION* message directly to its parent. The parent acknowledges the reception of this message with *PARENT_ACK*. We show in Figure 28 the state transition diagram of the TDMA slot assignment process.

Figure 29 illustrates the format of a message that is used for schedule exchange purposes, i.e. *SCHEDULE_ANNOUNCEMENT*, *SCHEDULE_CONFLICT*, *SCHEDULE_NOT_CONFLICT* and *SCHEDULE_NOTIFICATION*. A schedule packet consists of:

1. *src_ID* is the sender of the message,
2. *dest_ID* is broadcast if used by *SCHEDULE_ANNOUNCEMENT*, the destination's ID if used by *SCHEDULE_CONFLICT*/*SCHEDULE_NOT_CONFLICT*, the parent's ID if used by *SCHEDULE_NOTIFICATION*,
3. *neighbour_level* specifies whether a node is in one-hop or two-hop neighbourhood of the sender of *SCHEDULE_ANNOUNCEMENT*,
4. *slot_list* records the schedule,
5. *highest_slot* specifies the TDMA frame length, and
6. *neighbour_list* is a list of neighbours' ID.

Field	*type*	*src_ID*	*dest_ID*	*neighbour_level*	*slot_list*	*highest_slot*	*neighbour_list*
Field size (bytes)	1	2	2	2	2 x *num_slot*	2	2 x *num_neighbour*

Figure 29: Schedule packet format

We introduce an idea of broadcast slot, so a node can send a *SYNCHRONISATION* message to synchronise its children. A non-leaf node (except the sink) waits until all of its children inform it of their schedules before assigning:

1. one unicast slot to send its own data,
2. several unicast slots to forward its descendants' data, and
3. a broadcast slot to synchronise its children.

A node assigns a slot to itself by selecting the smallest available slot which is not used within its two-hop neighbourhood. This means the same slot can be used by two nodes that are separated by more than two hops away. The node also assigns several slots that are equal to the number of descendants it has to forward its descendants' data. For each forwarding slot, the node selects the smallest available collision-free slot. In addition, the node also selects a special broadcast slot to synchronise its children. This assigned schedule is then informed to the two-hop neighbourhood.

Each node executes the slot assignment until the *SCHEDULE_NOTIFICATION* message reaches the sink. The slot assignment phase ends when the sink receives *SCHEDULE_NOTIFICATION* messages from all of its direct children and assigns a broadcast slot to synchronise them. Figure 30 illustrates assigned transmit slots in a data gathering tree of six nodes.

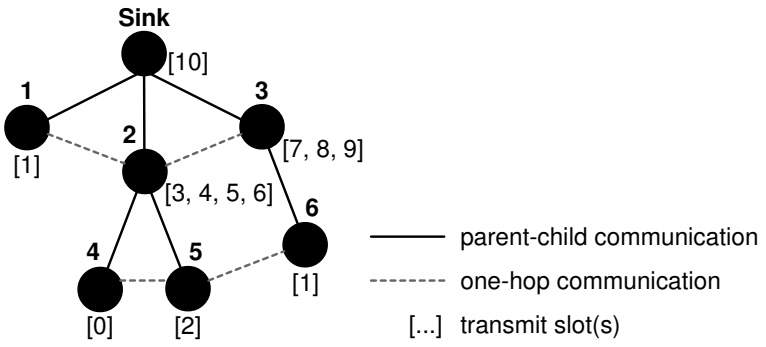


Figure 30: ER-MAC nodes' transmit schedules

The sink switches the communication mode to TDMA by sending the first *SYNCHRONISATION* message to all of its children, together with the information about the TDMA frame length. The purpose of propagating the TDMA frame length is to allow nodes in the network to keep the period of one TDMA frame length up to date. When a child receives the *SYNCHRONISATION* message, it switches its communication mode to TDMA and synchronises its children using its special broadcast slot. When all leaf nodes in the network receive a *SYNCHRONISATION* message, the whole network is switched to TDMA mode, synchronised, and each node in the network knows the exact duration of one TDMA frame.

4.3.3 Local Time Synchronisation

Time synchronisation is important in MAC protocols that adopt the schedule-based mechanism because nodes that have the same schedules for communication need to be active at the same time to transmit and receive messages. If the synchronisation messages are sent too often, they will incur a large amount of protocol overhead. If they are sent rarely, nodes will experience a large clock drift [40, 100].

We design ER-MAC with a local time synchronisation. Note that during the topology discovery of ER-MAC, each node discovers its parent and its children. Then, during the TDMA slot assignment, each node is assigned a special broadcast slot for synchronisation purposes. ER-MAC manages the local time synchronisation using a parent-children broadcast synchronisation, which is similar to the root-neighbours synchronisation of Flooding Time Synchronisation Protocol (FTSP) [76]. This simple mechanism is sufficient for our approach because each child only needs to have the same clock as its parent to ensure that the parent is in receive mode when it starts transmission and vice versa.

In the synchronisation slot, a parent broadcasts a *SYNCHRONISATION* message, which consists of:

1. *src_ID* is the parent's ID.

2. *current_slot* informs the current slot number to allow nodes that are not synchronised, such as new nodes, to synchronise themselves when they overhear this message.

3. *highest_slot* is the highest number of contention-free slot, that informs the TDMA frame length to allow nodes in the network to keep the period of one TDMA frame length up to date.

4. *clock* that informs the parent's clock to help children to synchronise their clock.

5. *hop_count* is the parent's hop count to the sink. This information helps a new node to select its prospective parent by choosing a parent node with the lowest hop count to the sink.

The format of a *SYNCHRONISATION* message is shown in Figure 31.

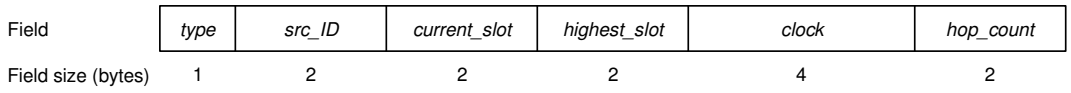


Figure 31: *SYNCHRONISATION* packet format

In ER-MAC, the local time synchronisation is performed once by each node that has child(ren) in each data gathering cycle to minimise clock drift. If a network has n nodes, there will be less than n *SYNCHRONISATION* messages sent during one data gathering cycle period because leaf nodes do not send these messages. This amount of overhead is fair and fixed regardless of the traffic rate. This scheme is more efficient than the scheme that requires a network-wide synchronisation before several contention-free slots, which is adopted by RRMAC [64]. There is also a traffic-based synchronisation, which is adopted by PMAC [127] and Z-MAC [97]. In the traffic-based synchronisation, each node sends one synchronisation message

according to the traffic rate in the network. With PMAC, a node sleeps for several time frames when there is no traffic in the network. It only sends a synchronisation message when it wakes up. With Z-MAC, a node sends a synchronisation message after sending 100 data packets. Compared to the traffic-based scheme, ER-MAC has more synchronisation overhead if the traffic load is light. However, the traffic-based scheme incurs large clock drift because of infrequent synchronisation. Additionally, if the traffic load is heavy, which is expected during an emergency monitoring, ER-MAC has less overhead. In the case of synchronisation error, an ER-MAC node can turn on its radio to overhear its neighbours' *SYNCHRONISATION* messages.

4.3.4 Priority Queue

ER-MAC uses two queues to separate high priority from low priority packets as shown in Figure 32. This multiple-queue system for sensor networks has been suggested in [9, 67, 37]. In our implementation of the priority queue, a packet is ordered based on its *slack*, i.e. the time remaining until the global packet deadline expires and is part of the packet header [33]. The format of ER-MAC's data packet is shown in Figure 33. The deadline is assigned by the WSN application to specify the desired bound on the end-to-end latency. A source node, which generates a data packet, initialises the slack with a deadline. The slack is updated at each hop by subtracting the queuing and transmission delays from it. To measure the queuing delay, a packet is timestamped when it is enqueued and dequeued. The queueing delay is the time difference between the enqueue and dequeue time. Then, to measure the transmission delay, a packet is timestamped when it is transmitted by a sender and received by a receiver. When a packet is re-transmitted, the slack is updated. The transmission delay is the time difference between the transmission time and the arrival time of a packet, given that the sender and receiver are locally synchronised.

We put the packet with the shortest slack in the front of the queue. Therefore, the shorter the slack, the sooner the packet should be transmitted. The rule of getting

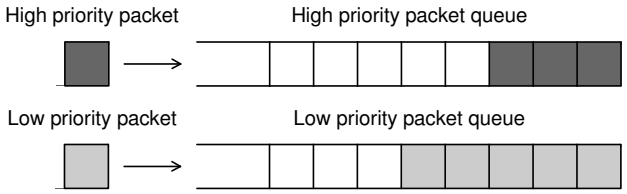


Figure 32: A pair of priority queues

Field	*type*	*src_ID*	*dest_ID*	*flag*	*priority*	*slack*	*timestamp*	*payload*
Field size (bytes)	1	2	2	1	1	4	4	

Figure 33: Data packet format

packets out of the queue is the high priority packets are transmitted first until the high priority queue is empty. If the high priority queue is empty, the packet in the front of the low priority queue is transmitted. A packet may be enqueued in a full queue. If this situation happens, we drop a packet with the shortest slack because it is most likely to miss its deadline and we assume that a packet that misses its deadline is useless. The consequence of this technique, however, is that messages from leaf nodes are dropped more frequently than others.

We also modify the implementation of the priority queue by considering fairness over the packets' sources, so the sink can have a balance of information from all sensor nodes. When the reporting frequency increases, a node may have lots of its own data in the queue. If the node always takes a packet from the head of the queue, it may happen that the node sends its own generated data more than its descendants' data. So, we modify our priority queue to transmit one packet from each descendant during one data gathering cycle period. We use an array, where the indexes correspond to nodes' ID, to record sources whose packets have been forwarded. We mark cell i in the array if node i's packet is dequeued. This array is reset every data gathering cycle. To dequeue a packet, we search through the queue to find a packet whose source has not been marked in the array. If such a packet exists, it will be dequeued and the source's cell is marked. If one packet

Algorithm 4: Fair-Dequeue

```
Input  : Queue, Source
Output: packet_to_send
 1:  packet_to_send ← null
 2:  if Queue is not empty then
 3:    for all packet in Queue do
 4:      if Source_{packet.source_address} = 0 then
 5:        packet_to_send ← packet
 6:      end if
 7:    end for
 8:    if packet_to_send = null then
 9:      packet_to_send ← Queue.head.packet
10:    end if
11:    Source_{packet_to_send.source_address} ← 1
12:  end if
13:  return packet_to_send
```

from every descendant has been forwarded, we take the packet from the head of the queue. This approach, however, has search time equivalent to the length of the queue in the worst case, because we may need to search the queue to the end for each transmitted packet. The pseudocode for this technique is presented in Algorithm 4.

4.3.5 MAC Prioritisation

The ER-MAC frame consists of contention-free slots with duration t_S each and a contention period with duration t_C as depicted in Figure 34. In each contention-free slot, except for the synchronisation slot, there are sub slots t_0, t_1, t_2 and t_3, which only appear in emergency mode for contention. Note that in the emergency mode, the period of $t_S - (t_0 + t_1 + t_2 + t_3)$ is sufficient to carry a packet and a sub slot is big enough to carry a MAC header (a source, a destination and a flag). However, the sub slots are not used in the normal mode, where a sender occupies a slot from the beginning of the slot and sleeps after transmitting a packet or at the end of the slot. We include a contention period at the end of each frame to support addition of new nodes. When a new node joins the network after a startup phase, it can use this contention period to find its parent and exchange schedules with its neighbours.

The exchange schedule process due to the addition of a new node, which will be discussed in Section 4.3.6, is carried out to the sink in each contention slot of data gathering cycle.

In normal monitoring, communication between sensor nodes follows the nodes' schedules. Every node sends its own data and forwards its descendants' data to its parent in collision-free slots. A node also has a special slot to broadcast synchronisation message or any messages to its children. To further conserve energy, a sender node turns off its radio if it has no data to send and a timeout forces a receiver node back to sleep if it does not receive any packets.

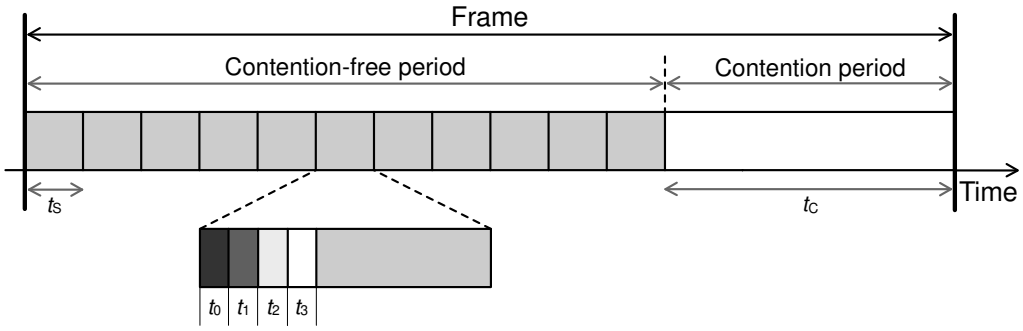


Figure 34: ER-MAC's frame structure

When fire is detected by some nodes' sensors, they change their MAC to emergency mode and set the emergency flag in their high and low priority packets. Note that only their parents can receive the packets with emergency flag because they are scheduled to wake up. To inform other neighbours of the emergency event, nodes that detect fire also broadcast *FIRE* messages to their one hop neighbours using their contention slots. The one-hop neighbours that receive the *FIRE* messages change their MAC to emergency mode so they can give up their transmit slots when needed by the nodes sensing the fire. The ancestors of the nodes caught in fire change their MAC to emergency mode when they receive data packets with emergency flag. These ancestors inform their one-hop neighbours to switch to emergency mode by broadcasting *FIRE* messages using their contention slots. The ancestors' one-hop neighbours change their MAC so they can give up their transmit slots when

needed by the nodes that are relaying emergency traffic. During the emergency situation, the whole network's MAC protocol is not switched in an instant, but hop by hop depending on the spread of the hazard. Nodes that do not participate in the emergency monitoring remain in the normal mode of ER-MAC.

Nodes change the behaviour of their MAC to emergency mode to achieve high delivery ratio and low latency by allowing contention in TDMA slots with the following rules:

1. An owner of a slot wakes up in the beginning of its own transmit slot. If it has a high priority packet to send, it transmits the packet immediately. If the owner has no high priority packet to send, it allows its one-hop neighbours with high priority packets to contend for the slot.

2. All non-owners of the slot wake up in the beginning of every slot to listen to the channel for possible contention or reception of packets. If a non-owner with a high priority packet senses no activities on the channel during t_0, it contends for the slot during t_1 by sending a *SLOT_REQUEST* message to the owner of the slot. The owner of the slot replies the request by sending a *SLOT_ACKNOWLEDGEMENT* to the requester.

3. The owner of the slot with low priority packets can only use its own slot if during $t_0 + t_1$ it does not receive any *SLOT_REQUEST* messages from its neighbours.

4. A non-owner with low priority packet can contend for the slot if during $t_0+t_1+t_2$ it senses no activities on the channel. Then, it contends for the slot during t_3 by sending a *SLOT_REQUEST* message to the owner of the slot. The owner of the slot replies the request by sending a *SLOT_ACKNOWLEDGEMENT* to the requester. Therefore, a node with low priority packets has a chance in every slot to contend for sending a packet.

A node that has switched to emergency mode may have neighbours that still operate in normal mode. Hence, it has to sense the channel before contending for its neighbours' transmit slot to avoid collision. If it does not sense any activities, it may contend for the slot, else it knows that the neighbour which is not in emergency mode is using its transmit slot. Moreover, to prevent a node sending an emergency packet to a sleeping parent, the first emergency packet is sent in a scheduled transmission slot. This will allow the ancestors of the node to switch their MAC protocol to the emergency mode when they receive the emergency packet. The delivery latency of the first emergency packet is however the same as in normal situation, but when nodes that involve in the emergency monitoring have switched their MAC protocol to the emergency mode, the latency of high priority packets is reduced.

A false alarm may happen in the network, where a node mistakenly thinks that it detects fire. If it happens, this node will inform its one-hop neighbours by sending a *FALSE_ALARM* message to change their MAC behaviour back to the normal mode. The ancestors of the node on the route to the sink that have already switched to emergency mode will change their MAC back to the normal mode if they do not receive any emergency packets after n data gathering cycle. They will also inform their one-hop neighbours regarding the false alarm.

4.3.6 New Nodes

The length of ER-MAC frame depends on the number of nodes in the routing tree. When a new node is added, the number of TDMA slots increases. ER-MAC supports addition of new nodes by utilising the contention slot at the end of each TDMA frame. When a new node is deployed, it has to listen to its neighbours' *SYNCHRONISATION* and data messages for at least one data gathering cycle. The *SYNCHRONISATION* message has several pieces of information that are useful to support addition of new nodes. The information about sender's ID and sender's hop count to the sink help the new node to select its parent. The new node will select a parent that has the lowest hop count to the sink. The *SYNCHRONISATION*

message also reports the current slot number, the highest slot number and the clock of the prospective parent to help the new node synchronises its clock and wait until the next contention slot to perform schedule exchange.

The slot assignment for a new node is similar to the slot assignment during the initial setup phase as described in Section 4.3.2, except that the new node takes the highest slot number incremented by one to be its slot number and the schedule exchange is performed in a contention slot. The new node generates a *SCHEDULE_ANNOUNCEMENT* message, appends its schedule and broadcasts the message to its one-hop neighbours. Nodes in its one-hop neighbourhood then rebroadcast this message to the two-hop neighbours. The new node has to wait until it receives no more *SCHEDULE_NOT_CONFLICT* messages from its two-hop neighbourhood after broadcasting the *SCHEDULE_ANNOUNCEMENT*. The new node then sends its assigned schedule in a *SCHEDULE_NOTIFICATION* message directly to its new parent. When the parent receives *SCHEDULE_NOTIFICATION* message from the new node, it acknowledges the new node as its child and adds the new node's transmit schedule to its receive schedule.

The parent then has to allocate one transmit slot to forward the new node's data and one slot to synchronise it if the parent has no children before. The transmit slot and the synchronisation slot are the new node's slot number incremented by one and two, respectively. The parent then performs schedule exchange during the next contention slot. The process of allocating new transmit slots because of the addition of the new node is carried out along the new node's routers toward the sink in each contention slot of data gathering cycle. It takes approximately $(l + 1) \times t$ seconds until the slot assignment reaches the sink since the new node is deployed, where l is the new node's hop count to the sink and t is one data gathering cycle period. Note that the one additional data gathering cycle is used by the new node to overhear its neighbours' *SYNCHRONISATION* and data messages prior to assigning its own schedule. The process of allocating new transmit slots because of the addition of a new node is illustrated in Figure 35.

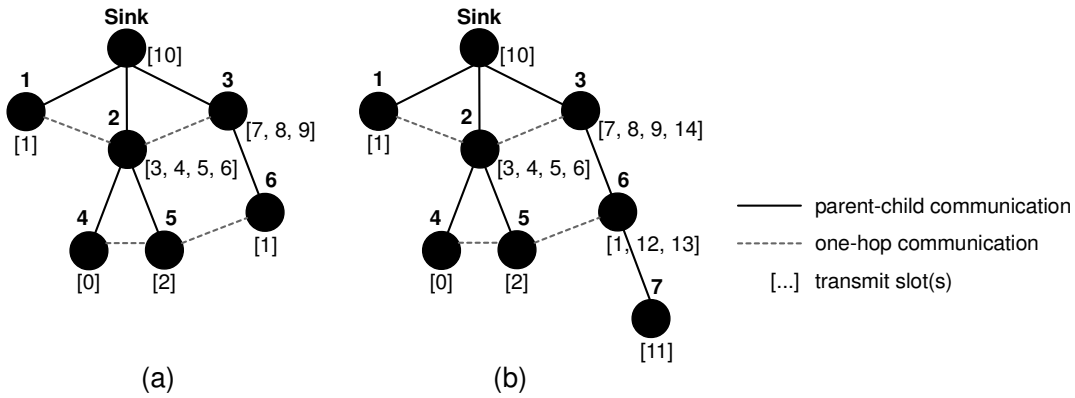


Figure 35: Addition of a new node, where (a) is the original network and (b) shows the network after node 7 is added

The addition of new slots lengthens the TDMA frame. Therefore, these changes must be informed to all nodes and they have to adjust their TDMA frame length simultaneously in the beginning of a data gathering cycle. The new node and the routers also start using their new allocated slots in the same data gathering cycle. This will prevent schedule clash where some part of the network has already changed its TDMA frame length while some other still use the old TDMA frame length. To apply the changes, a count down timer, set to be $l_{\max} \times t$ seconds, is piggybacked in the sink's *SYNCHRONISATION* message and is propagated to the whole network when a node synchronises its children. $l_{\max}$ is the highest hop count of the network. As the timer expires, all nodes simultaneously use the new schedules. The process of disseminating the new frame length proceeds until all nodes change their TDMA frame length and takes at most $l_{\max}$ data gathering cycle periods. Hence, the total time needed for a new node to operate in TDMA mode after it is deployed is $(l_{\max} + l + 1) \times t$ seconds.

The frame length inconsistencies, where some nodes use old frame length and some nodes use new frame length, are unlikely to happen because the synchronisation slots are collision-free. Moreover, it takes $l_{\max}$ data gathering cycle periods to disseminate the new frame length information. If a node does not receive a *SYNCHRONISATION* message due to temporary noisy links, it will receive it in the next period. If the links are permanently noisy, the node will find a new parent.

4.3.7 Dead Nodes

A node is dead if it runs out of battery or is destroyed by fire. We can also assume a node is dead if it cannot communicate with its parent or its children due to noisy links or obstacles. If a parent does not receive any data during all scheduled receive slots of a child after n consecutive data gathering cycles (user parameter), where n is usually greater than one to deal with temporary link failure, it assumes that the child is dead. The parent then removes the child from its children list. It also removes m scheduled receive slots that are associated with that child to prevent idle listening. If the child is the only child of that parent, it also removes the synchronisation's broadcast slot. Moreover, the parent also removes m transmit slots to forward that child and its descendants' data. The parent is then responsible to inform all the routers toward the sink to remove m receive slots associated with the removal of one of its children and m transmit slots from their schedules. This information is piggybacked on the data packet sent in the immediate transmit slot. All of the unused slots are then informed within two-hop neighbourhood in the contention slot.

When a node does not receive *SYNCHRONISATION* messages after n data gathering cycle from its parent, it may assume that its parent is dead. The orphan node then finds a new parent by following the same procedure as the new node deployment. In a contention slot, the orphan node will send its transmit slots' schedule in a *SCHEDULE_NOTIFICATION* message directly to its new parent, so the descendants of the orphan node do not need to rebuild their schedules. When the parent receives the *SCHEDULE_NOTIFICATION* from the orphan node, it acknowledges the orphan node as its child and adds the orphan node's transmit schedule to its receive schedule. The parent then assigns new transmit slots to forward its new child and new descendants' data. This schedule assignment is the same as the new node's assignment, except that there may be more than one slot that needs to be allocated because the orphan node may have children and descendants. To reuse some released slots, this schedule assignment will firstly search for

the smallest available slot, which does not conflict with the schedule of the node's two-hop neighbours.

4.3.8 Protocol Overhead

ER-MAC incurs higher protocol overhead at the beginning, i.e. during topology discovery and TDMA slot assignment phases. During this initial startup, ER-MAC communicates using CSMA/CA, where there are RTS/CTS/ACK in each transmission. After the initial startup, ER-MAC's protocol overhead during normal monitoring is only caused by *SYNCHRONISATION* messages, which are sent once each data gathering cycle by every node with child(ren). This amount of overhead is fixed regardless of the traffic rate and bounded by the number of nodes.

During emergency monitoring, besides *SYNCHRONISATION* messages, *FIRE* or *FALSE_ALARM*, *SLOT_REQUEST*, and *SLOT_ACKNOWLEDGEMENT* messages also contribute to the amount of protocol overhead. These four types of messages are generated only by nodes involved in emergency monitoring. Therefore, the amount of the overhead depends on those nodes. While *FIRE* or *FALSE_ALARM* messages are sent once every data gathering cycle in the contention slot, *SLOT_REQUEST* and *SLOT_ACKNOWLEDGEMENT* messages might be sent by several nodes in every TDMA slot for possible contention.

4.4 Evaluation of ER-MAC

By these experiments, we want to show that ER-MAC delivers low latency for high priority packets especially during emergency monitoring, it has fair packet delivery and nodes in non-emergency mode behave in an energy-efficient manner. In the simulation, we use the following metrics to measure the performance of ER-MAC:

1. ***Average energy consumption per node***. We want to show that ER-MAC is energy-efficient in normal situations.

2. ***Packet delivery ratio***. We want to show that ER-MAC has high packet delivery ratio, especially for high priority packets, for both normal and emergency situations.

3. ***Average per packet latency***. We want to show that the average per packet latency, especially for high priority packets, is reduced during an emergency situation.

4. ***Completeness of packets received***. We want to show that the sink can always have a balance of information from all sensor nodes in either normal situation or emergency situation.

We implemented ER-MAC in ns-2 [2]. Our simulation results are based on the mean value of five different network deployments that are simulated five times each using random seeds. The network consists of 100 nodes deployed within randomly perturbed grids. This is an approximation of manual deployments of sensor nodes, for example in a building layout. In the random perturbed grids, each node is placed in one unit grid square of 8 m $\times$ 8 m and the coordinates are slightly perturbed. This grid size is chosen in relation to the use of 10-metre transmission range, which is realistic for 0 dBm transmission power in an indoor environment [119]. The location of the sink was fixed at the top-left corner of the network. We randomly select up to n links and for each drop up to m packets, where m is large enough to model unreliable links. Moreover, for simplicity, we assume the links are symmetric. Our simulation parameters were based on Tmote sky hardware [4]. Table 8 presents our simulation parameters.

4.4.1 Protocol Comparison

We compared the performance of ER-MAC with Z-MAC, because this protocol has several similar characteristics with ours, such as hybrid designs and allowing contention in TDMA slots when the traffic load increases. We followed the Z-MAC ns-2

Table 8: ER-MAC and Z-MAC simulation parameters in ns-2

Simulation parameters	**Default value**
Transmission range	10 m
Transmit power (*txPower*)	52.2 mW
Receive power (*rxPower*)	59.1 mW
Idle power	59.1 mW
Sleep power	3 μW
Transition power	59.1 mW
Transition time	580 μs
Node initial energy	20,000 J (2 $\times$ AA batteries)
ER-MAC TDMA slot size	50 ms
ER-MAC TDMA sub-slot size	5 ms
Z-MAC TDMA slot size	50 ms
Z-MAC owner contention window size (T_{o})	8
Z-MAC non-owner contention window size (T_{no})	32

installation manual detailed in [5] and configured Z-MAC according to the default settings in [97]. Z-MAC's configuration is shown in the simulation parameters's table (Table 8). In addition, we use the same 10-metre transmission range as in ER-MAC's simulations. In each experiment, we simulated a data gathering for 300 seconds, where every node except the sink is a source node that generates packets with fixed intervals.

In the simulations, we compared the performance of ER-MAC with Z-MAC in terms of average energy consumption per node, packet delivery ratio, average per packet latency, and completeness of packets received at the sink. For ER-MAC simulations, we considered two network scenarios, i.e. no-fire and in-fire situations. In the no-fire situation, communication among nodes follows their TDMA schedules. However, in the in-fire situation, ER-MAC allows contention for the TDMA slots within one-hop neighbourhood if the owner of the slot has no data to send. To simulate the in-fire situation, we assume all nodes operate in emergency mode from the beginning of the simulation. For Z-MAC simulations, we forced Z-MAC to operate in either Low Contention Level (LCL) or High Contention Level (HCL) to model our no-fire and in-fire situations, respectively. Note that in LCL, any node can compete to

transmit in any slots, but in HCL only the owner of the current slot and their one-hop neighbours are allowed to compete for the slot. Our simulation results show that ER-MAC outperforms Z-MAC especially when the traffic load increases.

Figure 36 shows the average energy consumption per node during the simulations. ER-MAC nodes in both no-fire and in-fire situations consume less energy than Z-MAC nodes that operate in LCL and HCL modes. This is because in ER-MAC, the owner of the slot does not need to contend to access the channel if it has data to send. However, in Z-MAC, although the owner of the slot has priority to access the medium, it has to contend for the medium before sending its own data. The figure also shows that during the in-fire situation, ER-MAC nodes spend more energy than the no-fire situation, because they wake up in every slot for possible contention. The energy consumption of ER-MAC nodes during the in-fire situation is high when the traffic load is low (less than 0.1 packets/node/sec) because more nodes do not use their own transmit slots to send their data, but contend for their one-hop neighbours' transmit slots if the neighbours have no data to send. In other words, during the in-fire situation, the lighter the load, the more the possibilities for contention in the network.

We also extend our simulations by increasing the traffic load up to 1 packet/node/sec. The average energy consumption per node for the whole simulation is shown in Figure 37. In our simulation, the network reaches its peak load at around 0.2 packets/node/sec. Hence, the energy consumption of nodes above the peak load is stable as the nodes can only communicate using their own scheduled time slots even though they have more data to send in the queues. The possibility of contention above the peak load is also minimal because nodes always have data to send in their own slots.

In this simulation, we want to compare the delivery ratio of high and low priority packets. So, we force source nodes to generate the two kinds of packets at the same time. Figure 38 shows that ER-MAC's high priority packets achieve better delivery ratio than Z-MAC's packets and ER-MAC's low priority packets. In the

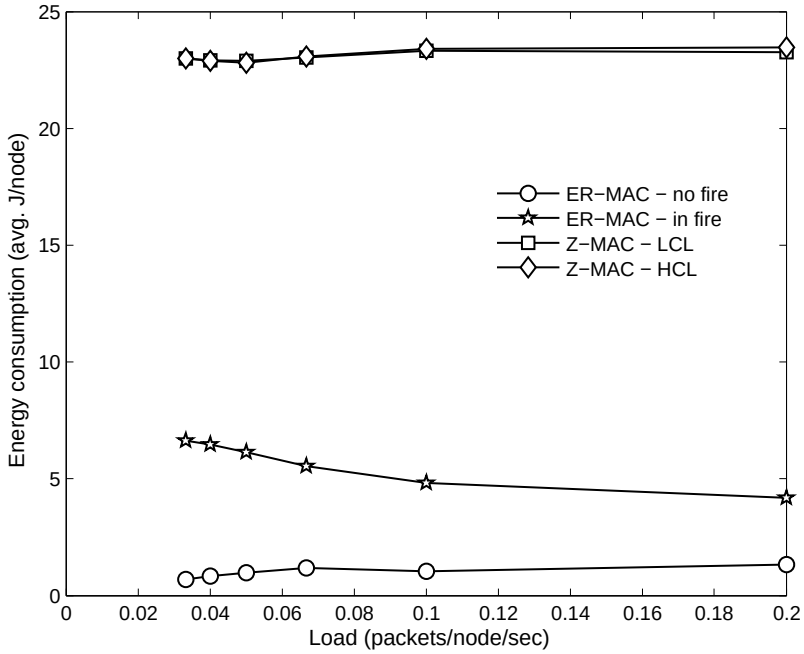


Figure 36: Energy consumption of ER-MAC versus Z-MAC

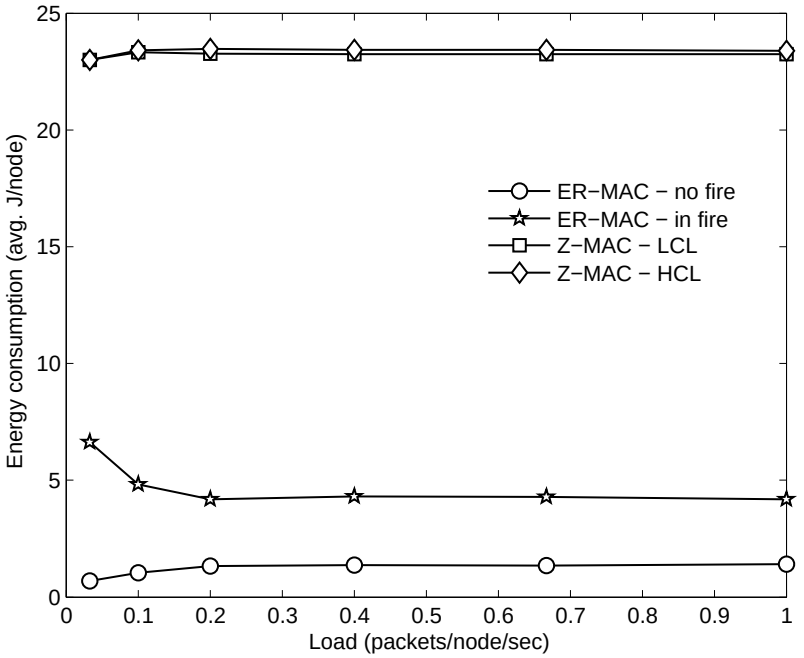


Figure 37: Energy consumption of ER-MAC versus Z-MAC for increasing load up to 1 packet/node/sec

figure, the lines for Z-MAC's low priority packets are hidden below the high priority. Z-MAC delivers the same delivery ratio for the two types of packets because it does not prioritise the high priority ones. When the traffic is very light, Z-MAC that operates in HCL mode achieves higher delivery ratio than ER-MAC because of data retransmissions when the packets are lost and the senders do not receive acknowledgements. On the other hand, ER-MAC does not acknowledge every data packets and so it does not retransmit lost data. Even though the delivery ratios of ER-MAC's high priority packets decrease when the traffic load increases, its delivery ratio in the in-fire situation is slightly higher than in the no-fire situation. This phenomenon is caused by contention in TDMA slots to prioritise the propagation of high priority packets during the emergency. However, the delivery ratio of ER-MAC's high priority packets does not change much from no-fire to in-fire because when nodes generate more traffic, the chance for contention is minimal. Figure 39 shows the packet delivery ratio when we increase the traffic load up to 1 packet/node/sec.

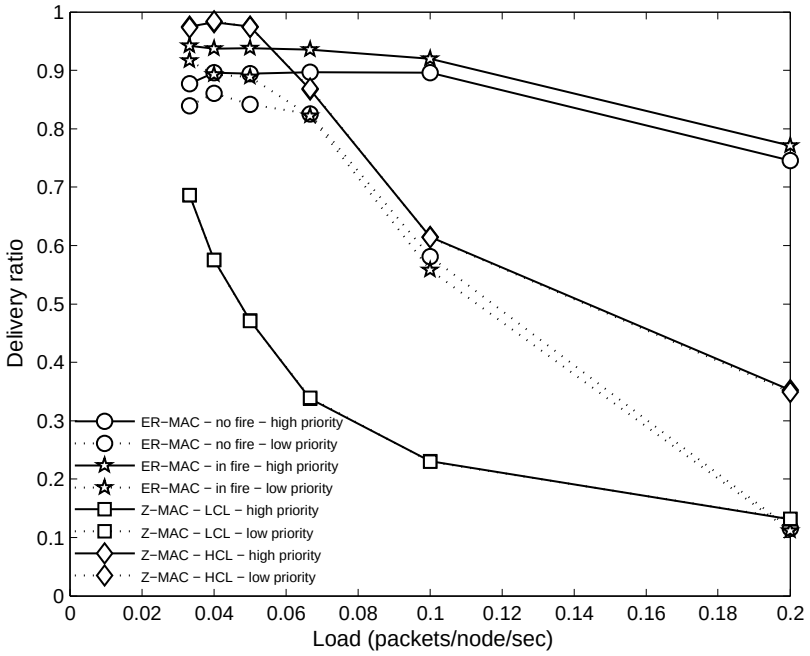


Figure 38: Delivery ratio of ER-MAC versus Z-MAC

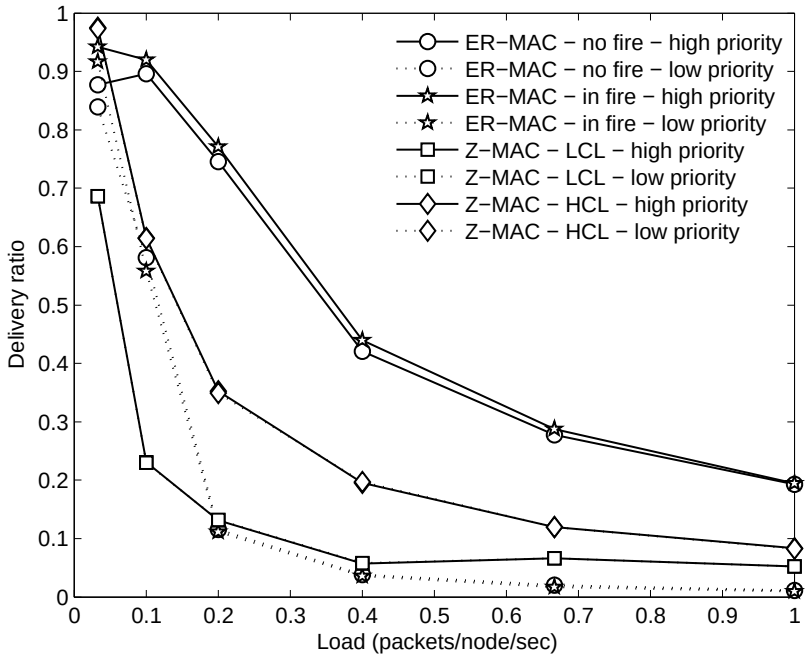


Figure 39: Delivery ratio of ER-MAC versus Z-MAC for increasing load up to 1 packet/node/sec

We want to show that the high priority packets have lower latency to reach the sink than the low priority packets. Figure 40 shows the average per packet latency of our simulations. ER-MAC's high priority packets generally have lower latency compared to Z-MAC's high priority packets. This is because ER-MAC maintains two priority queues that separates high priority packets from low priority ones and the high priority packets are always transmitted first until the queue is empty. On the other hand, Z-MAC only uses one queue and sends the high and low priority packets one after another. That is why the latency of Z-MAC's high and low priority packets almost have no differences. Moreover, ER-MAC prioritises high priority packets and so the latency of low priority packets is high. During the in-fire situation, ER-MAC's high priority packets' latency is reduced because nodes can propagate data quickly by contending for some unused slots.

Figure 41 shows the average per packet latency when we increase the traffic load up to 1 packet/node/sec. When the traffic load increases, the latency of ER-MAC's

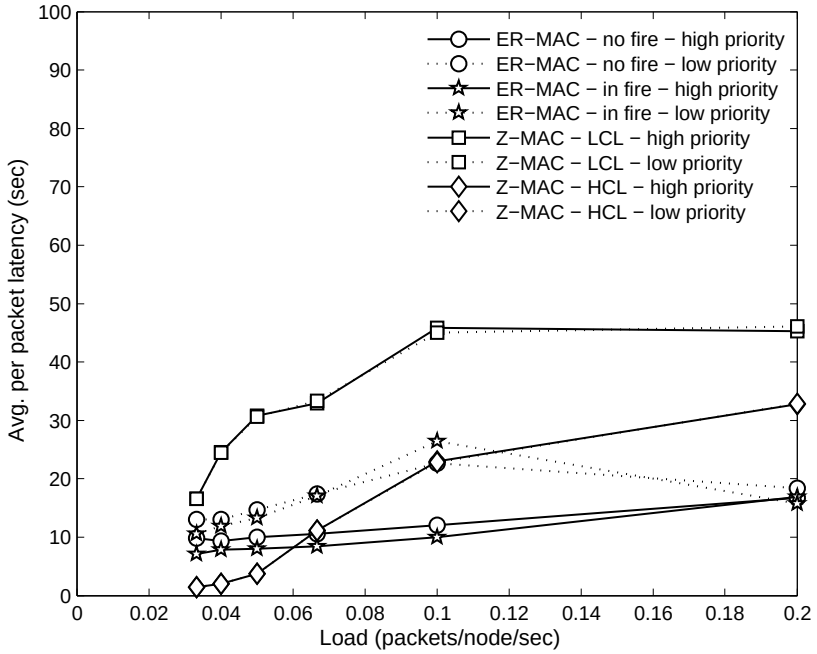


Figure 40: Latency of ER-MAC versus Z-MAC

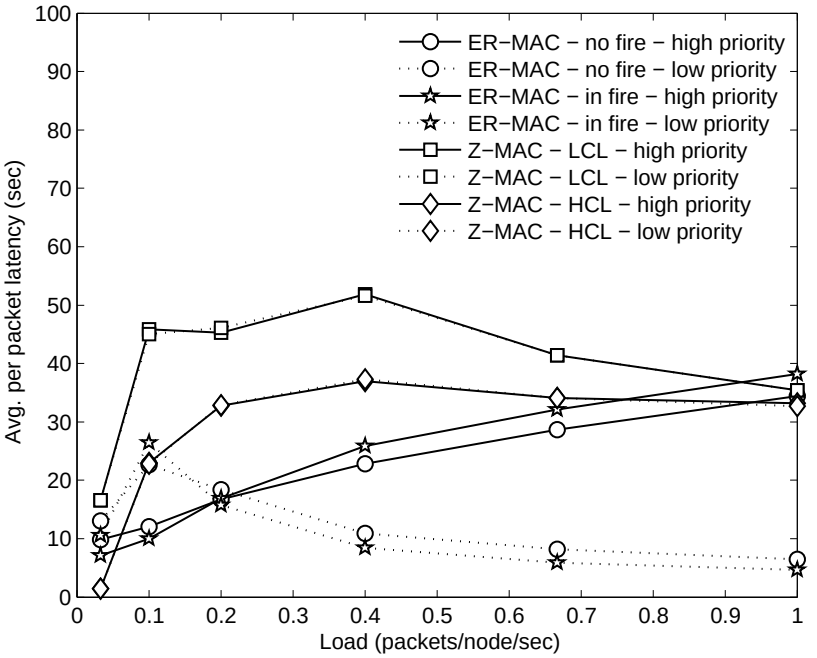


Figure 41: Latency of ER-MAC versus Z-MAC for increasing load up to 1 packet/node/sec

high priority packets rises. On the other hand, the latency of Z-MAC's packets and ER-MAC's low priority packets drops as we observe in the simulations that fewer packets are received at the sink and most of them are from nodes near it. This argument is validated by the low delivery ratio in Figure 39.

As explained in Section 4.3.4, we implement priority queues by considering fairness over the packets' sources. The reason behind this modification is we want the sink to have a balance of information from all sensor nodes in the network. Figure 42 shows the completeness of the packets received at the sink when the network reaches its peak load, i.e. 0.2 packets/node/sec. We measure the completeness as the percentage of packets received plotted against hop count. The graph shows that the completeness of ER-MAC's high priority packets for both no-fire and in-fire situations are higher than Z-MAC's packets and ER-MAC's low priority packets. This happens because of packet prioritisation and priority queue modification in ER-MAC to transmit one packet from each node during one data gathering cycle period.

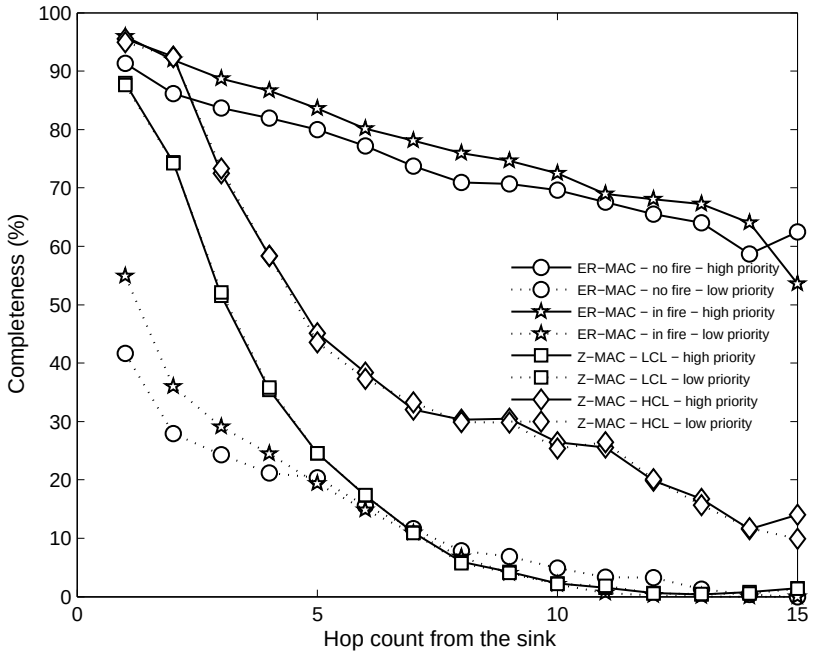


Figure 42: Completeness of ER-MAC versus Z-MAC

4.4.2 Behaviour When a Cluster of Nodes Detects Fire

We consider the situation when some nodes in a network detect fire. Nodes that detect fire become in-fire nodes. They change their MAC behaviour to emergency mode, set the emergency flag in each of their high and low priority packets and broadcast emergency messages to their one-hop neighbours during contention slots. The one-hop neighbours also switch to emergency mode but do not set the emergency flag in their data packets. When the ancestors of the in-fire nodes receive data packets with emergency flag, they change their MAC to emergency mode and broadcast emergency messages to their one-hop neighbours to change their MAC. Neither the ancestors nor their one-hop neighbours set the emergency flag in any of their packets. This situation is illustrated in Figure 43.

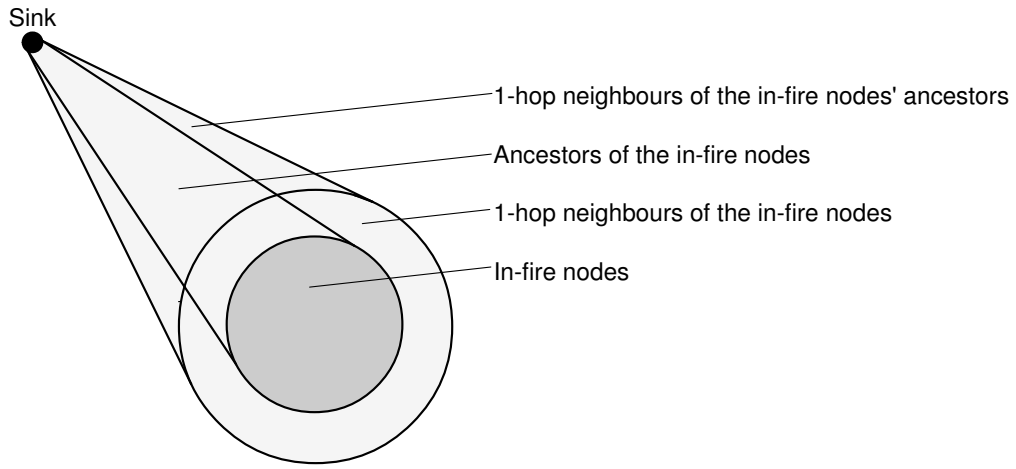


Figure 43: A cluster of nodes detects fire

We evaluate the performance of ER-MAC against Z-MAC when a cluster of nodes detects fire. For each simulation, we run a 500-second data gathering, where all nodes are the sources of high and low priority packets. They generate a constant 0.1 packets/node/sec traffic rate. 100 seconds after the simulation starts, a random location in the network is on fire. We choose five nodes, which are the closest nodes to the fire location, as in-fire nodes. The in-fire nodes double the traffic generation rate to 0.2 packets/node/sec and halve the packet deadline.

Figure 44 shows the average energy consumption per node during the 500-second simulations. The results reported at the 100^{th} second is when the network is not on fire. As the fire starts at the 100^{th} second, we start to plot the emergency monitoring results from the 200^{th} second. The simulation results show that ER-MAC is energy-efficient during this emergency monitoring as nodes consume less than one fifth of Z-MAC's energy consumption. The figure also shows that with ER-MAC, nodes that participate in the emergency monitoring, i.e. the in-fire nodes, the one-hop neighbours of the in-fire nodes, the ancestors of the in-fire nodes and the ancestors' one-hop neighbours, dominate the energy consumption of the network. Conversely, the rest of the network, which operates in the normal mode of ER-MAC, is very energy-efficient. Z-MAC does not distinguish between nodes that participate in the emergency monitoring and the normal monitoring. It switches from LCL to HCL mode if it detects heavy traffic loads. In addition, nodes that operate in the HCL and LCL modes of Z-MAC have been shown to consume almost the same amount of energy in Figure 36 and 37.

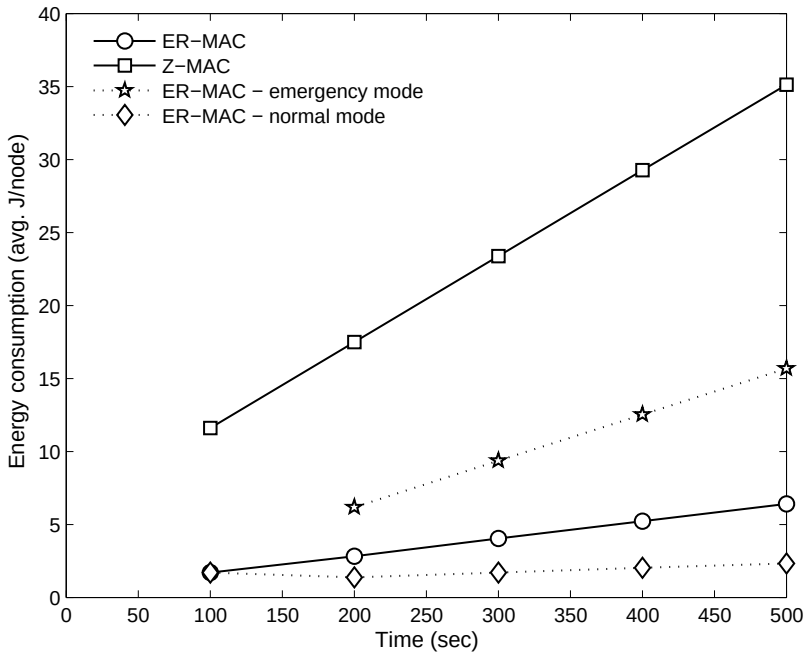


Figure 44: Energy consumption of ER-MAC versus Z-MAC when a cluster of nodes detects fire

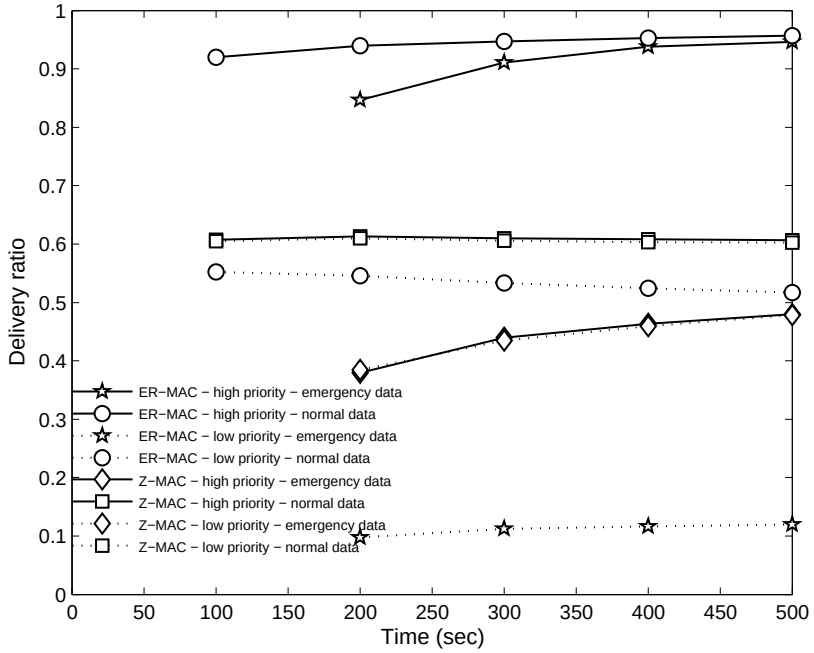


Figure 45: Delivery ratio of ER-MAC versus Z-MAC when a cluster of nodes detects fire

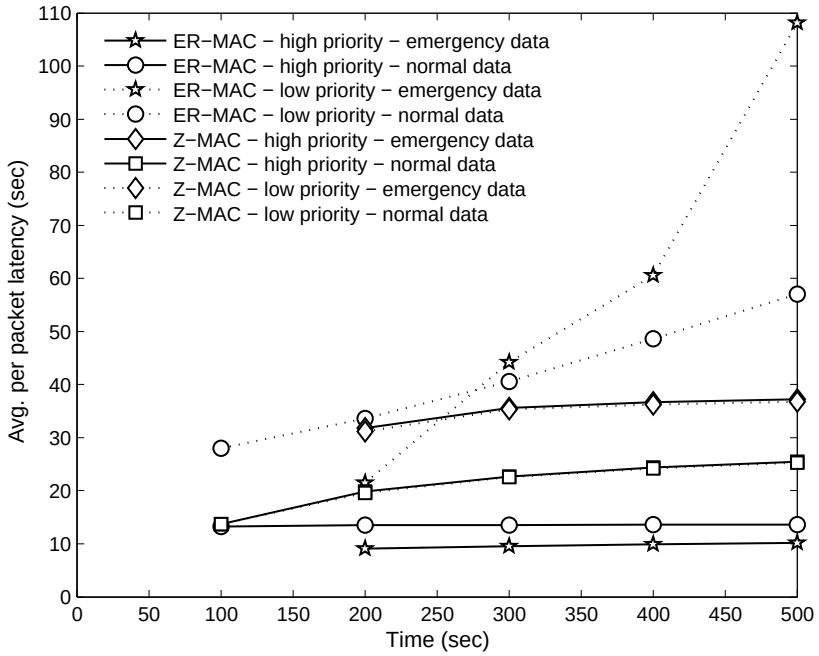


Figure 46: Latency of ER-MAC versus Z-MAC when a cluster of nodes detects fire

Figure 45 presents the delivery ratio of high and low priority packets with and without emergency flag. Recall that only in-fire nodes generate packets with emergency flag and their reporting frequency is twice as much as the normal data. Because of the ability to prioritise packets, ER-MAC achieves higher delivery ratio for emergency and normal high priority packets compared to Z-MAC, even though it sacrifices the low priority ones. ER-MAC also delivers the emergency high priority packets with the lowest latency as shown in Figure 46. This happens because emergency packets have shorter deadline than normal packets and so they are placed in the front of the queue. In our priority queue modification, the emergency packets are given the priority to be transmitted after one packet from each descendant of a node has been sent.

4.4.3 Behaviour Under Variable Traffic Load

In this simulation, we vary the traffic load during 500-second simulations. The traffic changes every 100 seconds. It jumps from 0.1 to 0.4 packets/node/sec, then drops to 0.1 packets/node/sec, and so forth. We vary the load in order to illustrate the changes in network conditions from no-fire to in-fire, then from in-fire to no-fire, and so on. When a node generates more traffic, it changes the MAC behaviour from the normal mode to the emergency mode. When it generates less traffic, it changes back to the normal mode. Figure 47, 48 and 49 show the comparison of ER-MAC against Z-MAC when the traffic changes over time in terms of average energy consumption per node, packet delivery ratio and average per packet latency, respectively. Overall, ER-MAC outperforms Z-MAC because it is more energy-efficient and its high priority packets have better delivery ratio and latency compared to Z-MAC's. In Figure 48 and 49, the delivery ratio and latency of Z-MAC's high and low priority packets overlap because Z-MAC only uses one queue and sends the high and low priority packets one after another. That is why the results are almost the same.

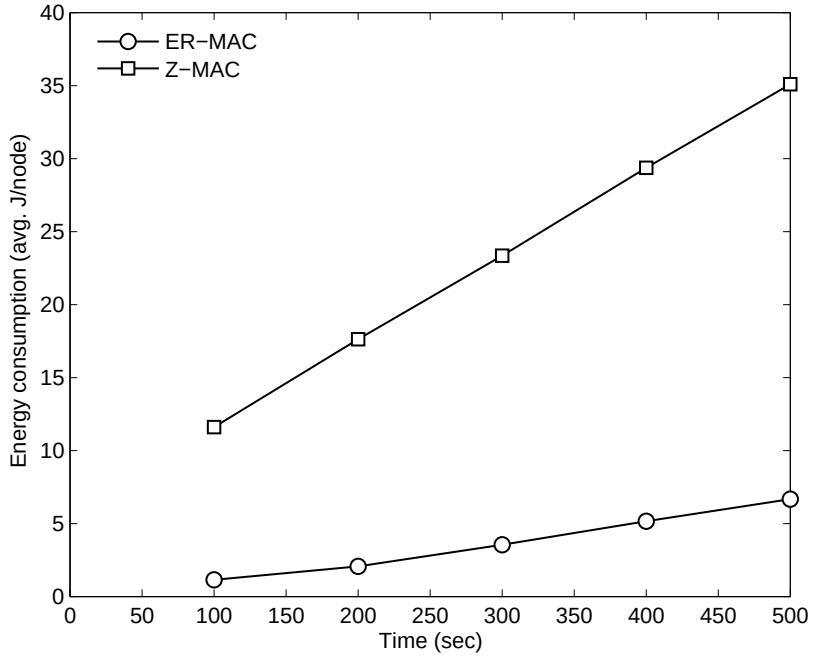


Figure 47: Energy consumption of ER-MAC versus Z-MAC under variable traffic load

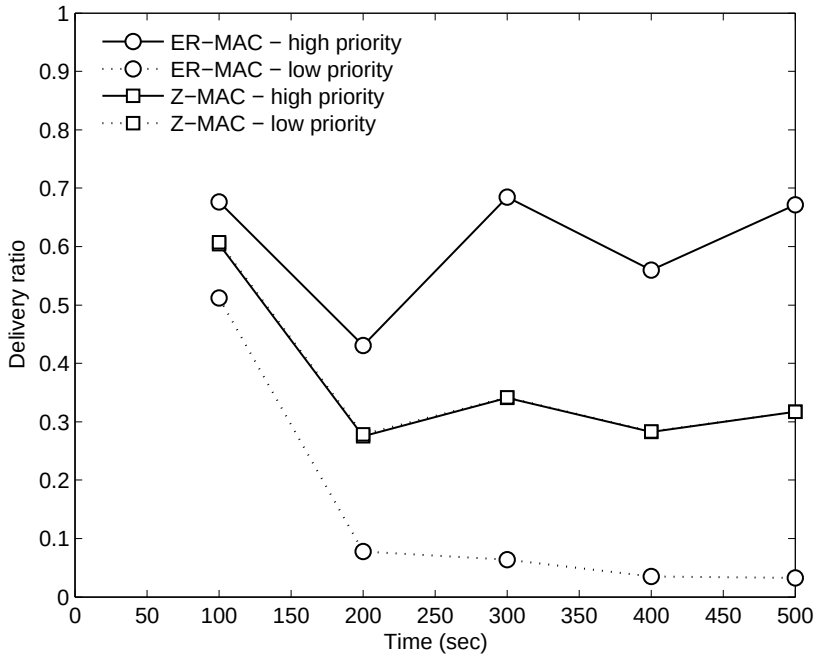


Figure 48: Delivery ratio of ER-MAC versus Z-MAC under variable traffic load

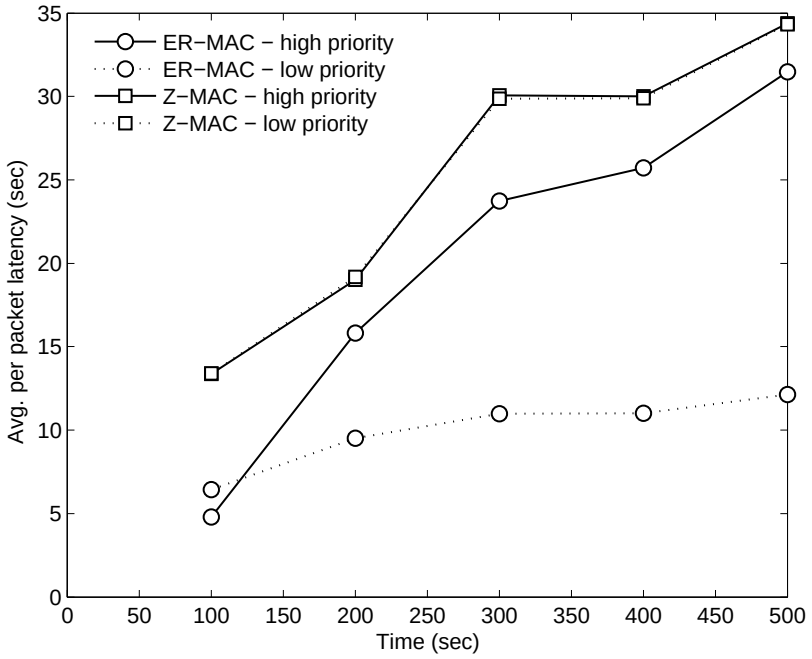


Figure 49: Latency of ER-MAC versus Z-MAC under variable traffic load

4.4.4 Behaviour When Topology Changes

We want to show that ER-MAC is topology adaptive by simulating networks while sensor nodes are failing. In the simulation, we increase the number of dead nodes from one to five and calculate the average energy consumption and time needed to reconfigure the network. The energy consumption to reconfigure the network is the amount of energy spent by orphan nodes to find their new parents and to announce new schedules in contention slots. The network reconnectivity latency is calculated from the time a node knows that its parent is dead until it uses its new TDMA schedules. In this simulation, a node is considered dead if after two data gathering cycles, its parent and children do not receive any packets from it. These simulation results are depicted in Figure 50. The amount of energy spent by a node to find a new parent is very small, i.e. less than 0.000125% of its initial energy. The reconnectivity latency slightly increases when more nodes die because the path length of an orphan node to the sink may be lengthened when it finds a new parent.

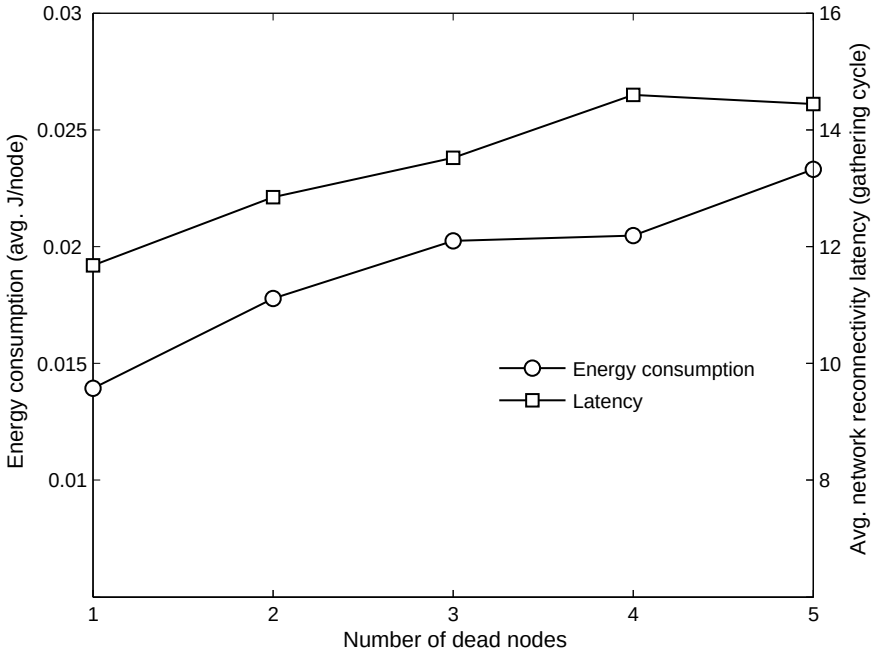


Figure 50: Energy consumption and latency of ER-MAC for network reconnectivity when some nodes die gradually

We also simulate the situation where several nodes die simultaneously. The simulation results are depicted in Figure 51, where we increase the number of dead nodes from 5 to 20. The energy consumption and latency to reconfigure the network decrease when the number of dead nodes goes over 15 because the network gets partitioned as the number of failed node increases. Hence, we only measure the energy expenditure and time to reconfigure the network from the remaining nodes that still form a connected network to the sink.

4.4.5 Behaviour Using Different Topologies

We want to evaluate the performance of ER-MAC by considering different topologies. The first topology, as shown in Figure 52, is a 100-node network which is easy to partition if a single node fails. The second one is the same network with five relay nodes as shown in Figure 53. The addition of five relays, i.e. node 100–104 in the network makes the network more robust against a single point of failure. At

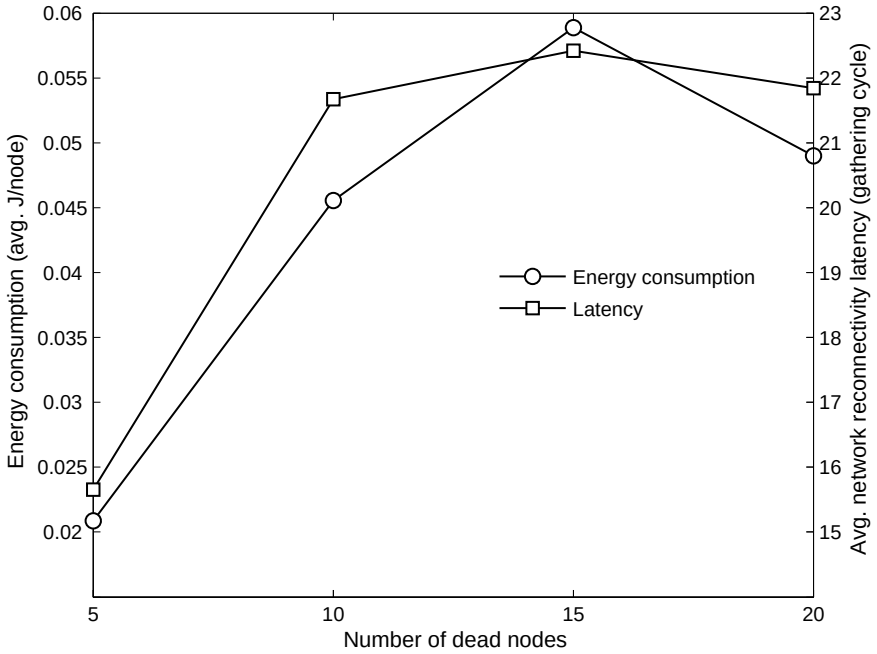


Figure 51: Energy consumption and latency of ER-MAC for network reconnectivity when some nodes die simultaneously

this stage, we only want to assess ER-MAC's performance in using a more robust topology, i.e. a topology with relay nodes. We will discuss the relay deployment problem to improve the network robustness in the subsequent chapters.

In the experiment, we simulate five runs of 1000-second data gathering on each topology using 0.03 packets/node/sec load, i.e. each sensor node generates two packets (high and low priority) every 60 seconds. Note that the relay nodes in the second topology do not generate traffic, but only forward sensor nodes' data. We evaluate ER-MAC while the network is in both no-fire and in-fire conditions. We place the sink at node 0's position and turn node 53 off during the simulation after the setup phase.

In the original topology, when node 53 fails we lose a significant portion of the network that consists of 31 sensor nodes. In the topology with relays, the network remains connected after the failure of node 53 and all nodes still relay traffic from the 31 sensor nodes toward the sink. Therefore, in both no-fire and in-fire situations,

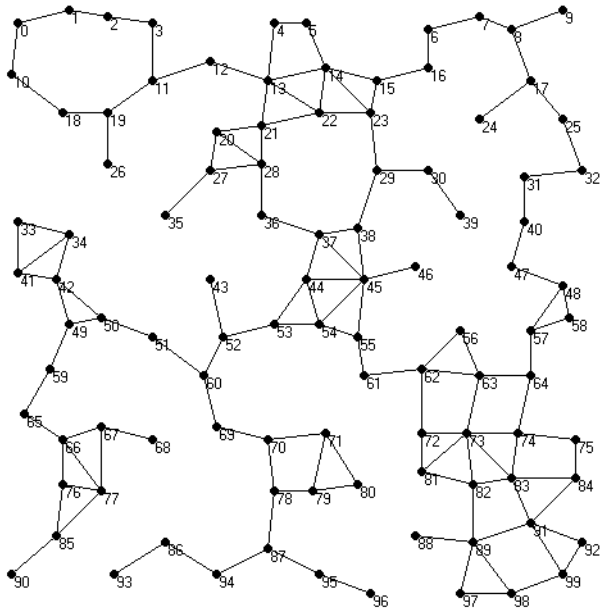


Figure 52: An example of a 100-node network which is easy to partition

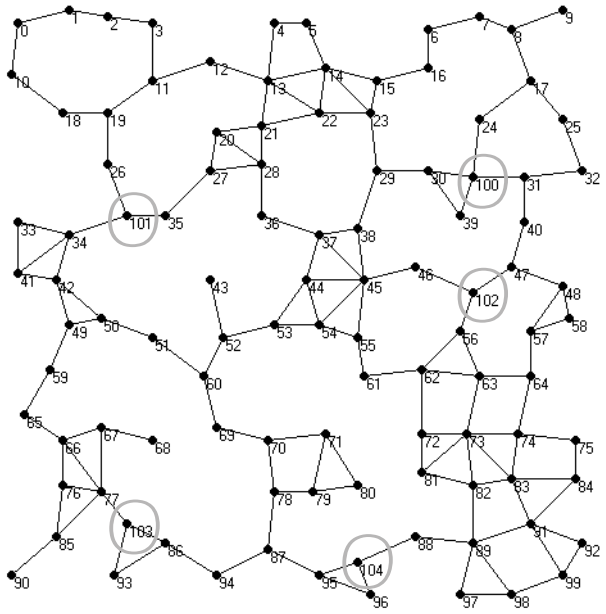


Figure 53: An example of a more robust network with five relay nodes: 100–104

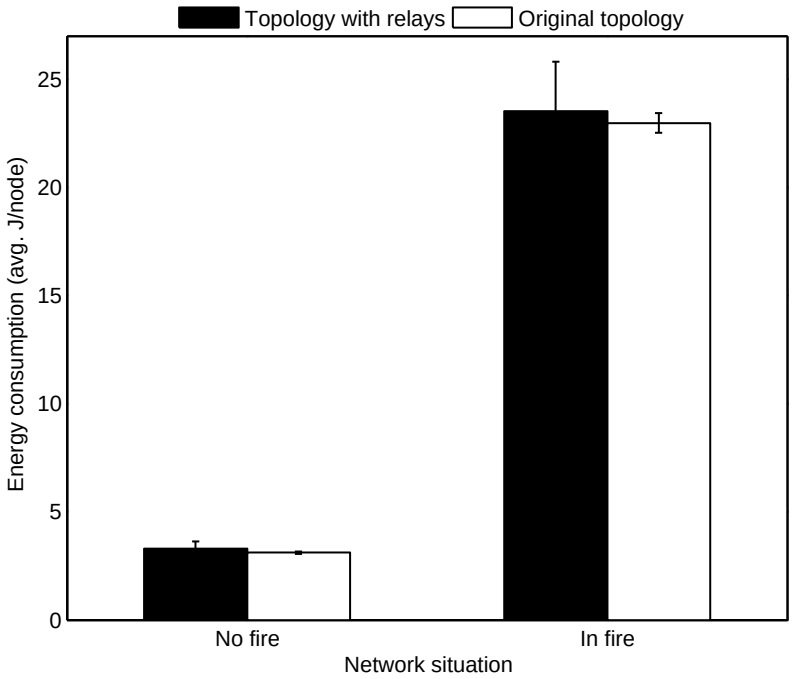


Figure 54: Energy consumption of ER-MAC using different topologies

they consume slightly more energy than the nodes in the original topology as shown in Figure 54. When the network is partitioned, the delivery ratios of both high and low priority packets of the original topology decrease as illustrated in Figure 55. The topology with relays, on the other hand, has better delivery ratios as the network is still connected after the failure of node 53. Figure 56 shows the delivery latency, where the topology with relays has lower end-to-end latency because the addition of five relays into the network shortens some sensor nodes' paths toward the sink.

4.4.6 Behaviour Using Different Sink Positions

We use different sink positions to investigate the performance of ER-MAC when the path lengths from sensor nodes to the sink are shortened. We use the topology in Figure 52 and run the simulation using two sink positions: at the top-left corner of the network (node 0's position) and in the centre of the network (node 44's position). When we move the sink to the centre of the network, we reduce the path

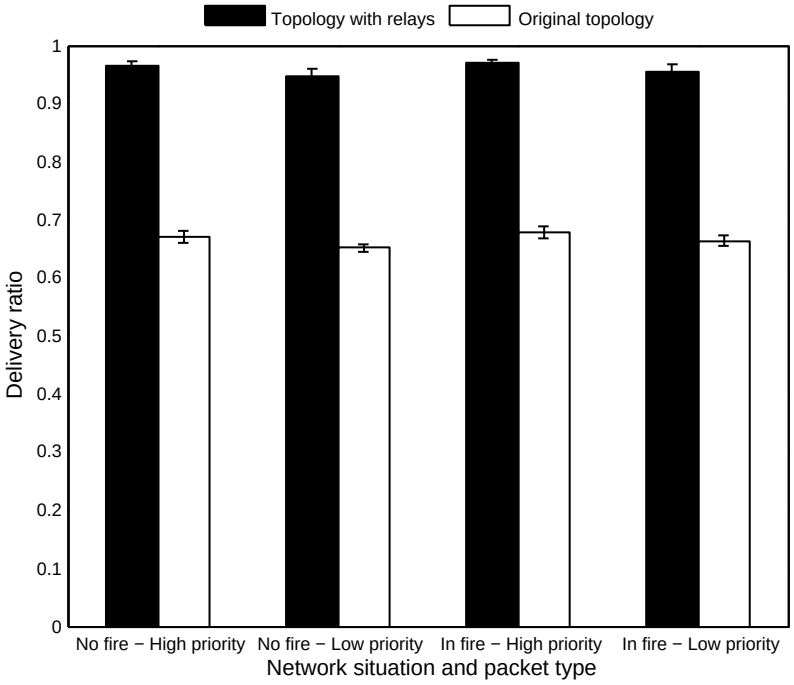


Figure 55: Delivery ratio of ER-MAC using different topologies

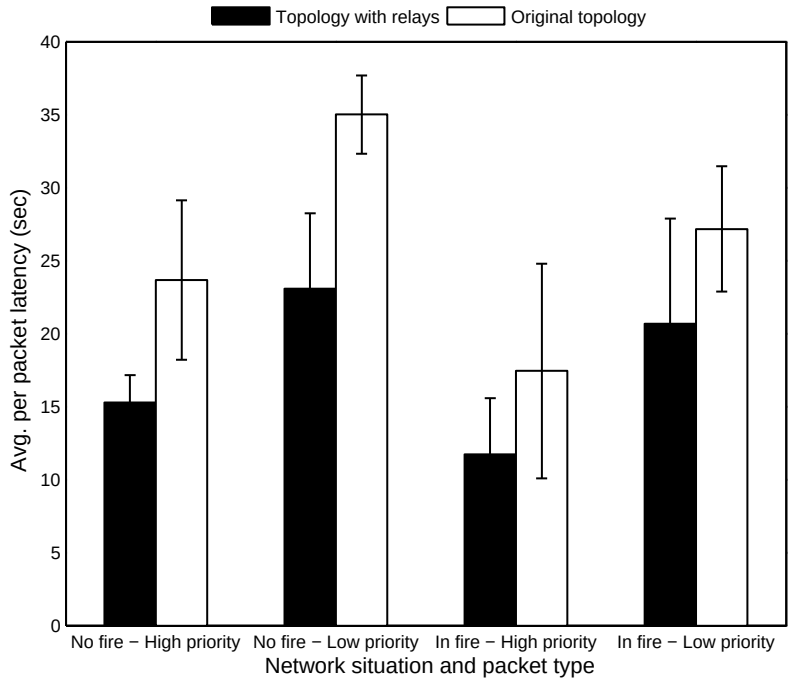


Figure 56: Latency of ER-MAC using different topologies

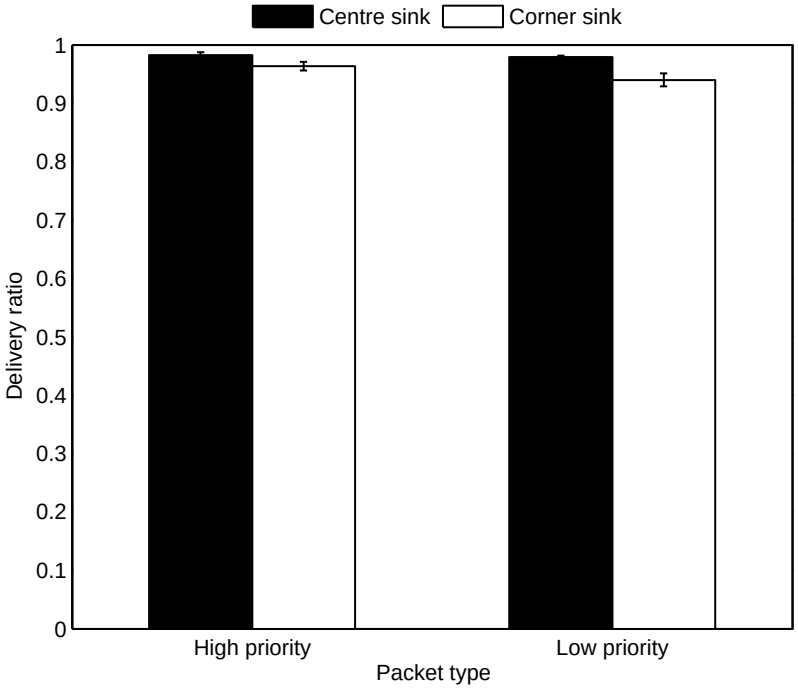


Figure 57: Delivery ratio of ER-MAC using different sink positions

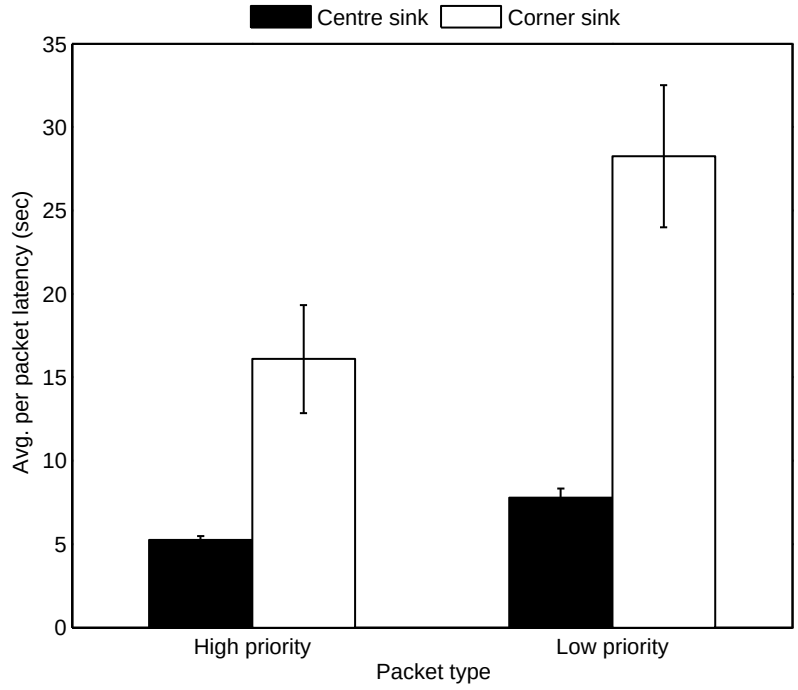


Figure 58: Latency of ER-MAC using different sink positions

length of distant nodes to reach the sink. Figure 57 and 58 show the delivery ratio and latency using each sink position, which are the average of five runs of data gathering simulation. The network gives a better performance when the sink is in the centre, because packet delivery ratio and the average per packet latency are influenced by the diameter of the network.

4.5 Conclusion

In this chapter, we present ER-MAC, a hybrid MAC protocol for emergency response WSNs with flexibility to adapt well to traffic and topology changes. ER-MAC schedules collision-free slots, so during the normal monitoring, nodes only wake up for their scheduled slots, but otherwise sleep to save energy. During an emergency, nodes that participate in the emergency monitoring change their MAC behaviour by allowing contention in each slot to achieve high delivery ratio and low latency, but have to sacrifice energy efficiency. ER-MAC is designed to prioritise high priority packets. It also offers a synchronised and loose slot structure, where nodes can modify their schedules locally. Our ns-2 simulation results demonstrate the scalability of ER-MAC and show that ER-MAC achieves higher delivery ratio and lower latency at low energy consumption compared to Z-MAC. We also show in our experiment that the performance of ER-MAC is greatly influenced by network topologies. ER-MAC gives a better performance on topologies that are not easy to partition and have shorter path length to the sink.

Network security is important in this emergency response application to prevent an attacker from switching the network into emergency mode as often as possible to deplete nodes' energy and flooding the network with high priority packets to fill in the queue with bogus data. The security aspect is out of scope of this thesis, but might be addressed using techniques, such as anomaly detection [92], sensor node behaviour profile modeling [117], and malicious node detection scheme [83]. In this thesis, we do not incorporate dynamic link estimation into ER-MAC. However, in

practice we can adapt one of the techniques from the literature, such as using the Received Signal Strength Indicator (RSSI) [107].

Chapter 5

Fault-Tolerant Relay Deployment for k Vertex-Disjoint Paths

5.1 Introduction

Ensuring that Wireless Sensor Networks (WSNs) are robust to failures requires that the physical network topology will offer alternative routes to the sinks. This requires sensor network deployments to be planned with an objective of ensuring some measure of robustness in the topology, so that when failures do occur, routing protocols can continue to offer reliable delivery. In the WSN deployment planning process, which usually includes requirements gathering, site survey, topology design and optimisation, our research focuses on the topology design and optimisation phases. Our contribution is a solution that enables fault-tolerant WSN deployment planning by judicious use of a minimum number of additional relays. We define a WSN to be robust if at least one route with an acceptable length to a sink is available for each remaining sensor node after the failure of up to $k-1$ nodes.

In this chapter, we define a novel problem for increasing WSN reliability by deploying a number of additional relays to ensure that each sensor node in the initial design has k length-bounded disjoint paths to one or more sinks. We define the

single-tiered, constrained partial fault-tolerant relay placement problem for k disjoint paths with a length constraint for WSNs with data sinks, which we call the Additional Relay Placement (ARP) problem. In single-tiered networks, all nodes can forward packets from other nodes. In constrained problems, relays can only be placed at candidate locations due to, for example, physical obstacles. Partial fault-tolerance requires k-connectivity only between every sensor node. We choose to focus on single-tiered networks as this is most common in the research literature and for published WSN deployments. We assume the constrained approach for relay locations, which we believe is more reasonable for real-world deployments. We also assume partial fault-tolerance, reflecting the fact that relays are deployed for connectivity only and do not have a sensing role. We assume that we are given a pre-planned WSN with a connected finite set of sensors and one or more sinks. We make no assumptions on the geographical or physical properties of the area in which the WSN is to be deployed, but we assume a set of possible locations for relays, and a connectivity graph, showing the set of feasible links between nodes.

We present two centralised algorithms to be run during the initial topology planning, i.e. prior to network deployment and operation, to solve this problem. *Counting-Paths* is a heuristic algorithm that counts the number of disjoint paths from each sensor node and finds the shortest disjoint paths to sinks. *Greedy Randomised Adaptive Search Procedure for Additional Relay Placement* (GRASP-ARP) is a local search algorithm that uses Counting-Paths to minimise the number of relays that need to be deployed. Specifically, the contributions of this chapter are:

1. The basic Counting-Paths algorithm uses a maximum flow algorithm, such as Ford-Fulkerson [43], to count the number of disjoint paths from sensors to sinks and to find the actual k shortest disjoint paths. For each sensor node, a set of disjoint paths to the sinks is sought, where:

   (a) the sum of the lengths is minimal, because we want to find for the shortest disjoint paths, and

(b) the length difference between the shortest path and the longest path is minimal, because we do not want a huge difference in alternative paths.

It is formulated as minimise $\sum_{i=1}^{k} l_i + (l_{\max} - l_{\min})$, where k is the number of disjoint paths, l is a path's length, $l_{\min}$ is the shortest length, and $l_{\max}$ is the longest length of the disjoint paths.

2. A dynamic programming variant of the Counting-Paths algorithm to count the number of disjoint paths from sensors to sinks and to find k shortest disjoint paths to k neighbours that already have k disjoint paths. If we are not interested in global routing paths during the deployment planning, it is not necessary for us to discover the actual paths, but only the number of disjoint paths and the neighbours that have k disjoint paths. The algorithm cannot find the actual paths because it does not store the complete paths.

3. GRASP-ARP is a local search algorithm that uses Counting-Paths for the single-tiered constrained partial fault-tolerant relay placement problem for k disjoint paths. It uses the concept of the GRASP algorithm to deploy a minimum number of additional relays at the possible candidate locations.

The rest of this chapter is organised as follows. We present Counting-Paths in Section 5.2 and its simulation results in Section 5.3. We show that Counting-Paths runs faster than the closest approaches that we compared with and is able to identify the maximum k such that a node has k disjoint paths. In addition, its dynamic programming variant improves on the runtime. We introduce GRASP-ARP in Section 5.4 that ensures length-bound with the basic Counting-Paths, but runs faster with the dynamic programming variant. We demonstrate empirically in Section 5.5 that it finds solutions requiring 35% fewer additional relays for small values of k compared to the closest approach from the literature. We also show that GRASP-ARP scales better, finding solutions in reasonable time for problems with hundreds of nodes. This represents a quantifiable improvement, making it possible to compute cost-effective WSN designs that offer an assured level of reliable delivery.

5.2 Counting-Paths

Counting-Paths is a heuristic algorithm to count the number of disjoint paths from a node. It utilises the Ford-Fulkerson [43] maximum flow algorithm, which is described in Appendix A.2, to find the actual disjoint paths. In each of its iterations, Counting-Paths finds the shortest path from a source node to a sink using the breadth first search technique. Without graph modification, Ford-Fulkerson can only discovers edge-disjoint paths [65] because if the capacity of all edges is one unit, Ford-Fulkerson's paths will not share a common edge, but may share common vertices. Therefore, before we find the second shortest path, we need to modify the original graph by using a vertex-splitting technique as is used in the algorithm proposed by Bhandari [19]. Vertex-splitting along the paths that have been discovered can exclude all possible paths that intersect them. To count the number of disjoint paths for all nodes in the network, we propose a dynamic programming variant of Counting-Paths, where we start counting the paths from sensor nodes closer to the sink. Dynamic programming solutions to problems are solutions to simple subproblems in a recursive fashion [86, 34]. This scheme speeds up the counting process for the entire network.

Existing disjoint paths algorithms by Torrieri [113] and Bhandari [19] are similar to Counting-Paths, which is designed to discover the shortest disjoint paths. However, Counting-Paths, which uses Ford-Fulkerson with the breadth first search technique, has lower time complexity than the two algorithms. Breadth first search has $O(|V|+|E|)$, which is slightly better than Bhandari's with Dijkstra's $O(|V|^2)$ and Torrieri's polynomial time. The time complexity of the Ford-Fulkerson algorithm is $O(|E|f)$, where f is the maximum flow in the graph. When we want to find k disjoint paths using Ford-Fulkerson, the time complexity becomes $O(|E|k)$. Moreover, with the dynamic programming variant, we can further reduce the time complexity.

We will discuss the problem of finding disjoint paths by firstly presenting the basic Counting-Paths algorithm to solve the single source – single sink problem. In this

problem, we want to check whether or not a node has k disjoint paths to a sink. Then, we will present the dynamic programming variant of Counting-Paths to solve the multiple sources – single sink problem. After that, we will discuss the variations of the algorithm to solve cases with multiple sinks.

5.2.1 Single Source – Single Sink Problem

In *finding k shortest disjoint paths for the single source – single sink problem*, given a graph $G=(V,E)$, we check if a source $s \in V$ has k disjoint paths to a destination $t \in V$, $t \neq s$, by finding the k shortest disjoint paths, if they exist, from s to t, where $\sum_{i=1}^{k} l_i + (l_{\max} - l_{\min})$ is minimised. k denotes the number of disjoint paths, l is the length of a path, $l_{\min}$ is the shortest length, and $l_{\max}$ is the longest length of the disjoint paths. If $k=\infty$, we find all possible disjoint paths from s to t.

Counting-Paths uses the Ford-Fulkerson method, which is iterative. It starts by giving an initial flow of value zero. Then at each iteration, the flow value is increased by finding an augmenting path from the source to the sink along which we can send more flow. A path P has a cost attribute, denoted as $cost(P)$. The cost of pushing a flow along an edge is defined as one unit of cost to send one unit of flow from a vertex to one of its adjacent vertices. A *path cost* is the total amount of cost to push each flow along each edge on a path. The cost is subtracted with a flow if the direction of the path is opposite to the direction of the flow. Given a flow network and a flow, the residual network consists of edges that can admit more flow. Formally, if we have a flow network G with a source and a sink, the residual network G_{res} is the network with residual capacity $capacity_{\text{res}}(v,w) = capacity(v,w) - flow(v,w)$.

A flow network is a directed graph, where each directed edge has a stated capacity. In our scenario for k disjoint paths, the WSN topology is an undirected graph and the total capacity of each edge is one. Therefore, we need a slight modification of the Ford-Fulkerson method to work with our specific network requirements. We also utilise the vertex-splitting technique [43] as is used in Bhandari's algorithm [19] to

Algorithm 5: Counting-Paths

Input : G, s, t, k
Output: P_i, $\forall i = 1, \ldots, k$

```
1:  for i ← 1 to k do
2:    if i > 1 then
3:      Split vertices on the shortest paths except s and t
4:      Modify the residual network G_res
5:      Replace external edges connected to the vertices on the shortest paths
        except s and t
6:    end if
7:    if there exists a path P_i from s to t in G_res then
8:      Push flow along P_i towards t
9:    end if
10:   if i > 1 then
11:     Remove overlapping edges
12:   end if
13: end for
14: return P_i, ∀i = 1, ..., k
```

exclude all possible paths that intersect the previously discovered paths. Because we utilise the vertex-splitting technique, we need to modify the breadth first search algorithm so that it is able to find the shortest path with the least path cost. This modification will be explained later in the description of Counting-Paths.

We present the basic Counting-Paths algorithm in Algorithm 5 to solve the k shortest disjoint paths for the single source – single sink problem. Counting-Paths is a combination of Ford-Fulkerson with breadth first search and the vertex-splitting technique. It takes as input a graph G, a source s, a destination t, and the number of disjoint paths sought k. The details of the steps are given below and an example to illustrate the steps when we explain the algorithm is shown in Figure 59.

Suppose we have an input network as depicted in Figure 59(a). We want to find two shortest disjoint paths from the source s to the sink t. An undirected edge (v, w) in the residual network shows that a directed edge may exist either from v to w or from w to v with the total capacity of one. For example, the first augmenting path found is $P_1 = \{s, a, c, t\}$ as shown in Figure 59(a). The flow is pushed from s to t along P_1 as shown in Figure 59(b). We follow Algorithm 5, which is described in details below, to find the second disjoint path.

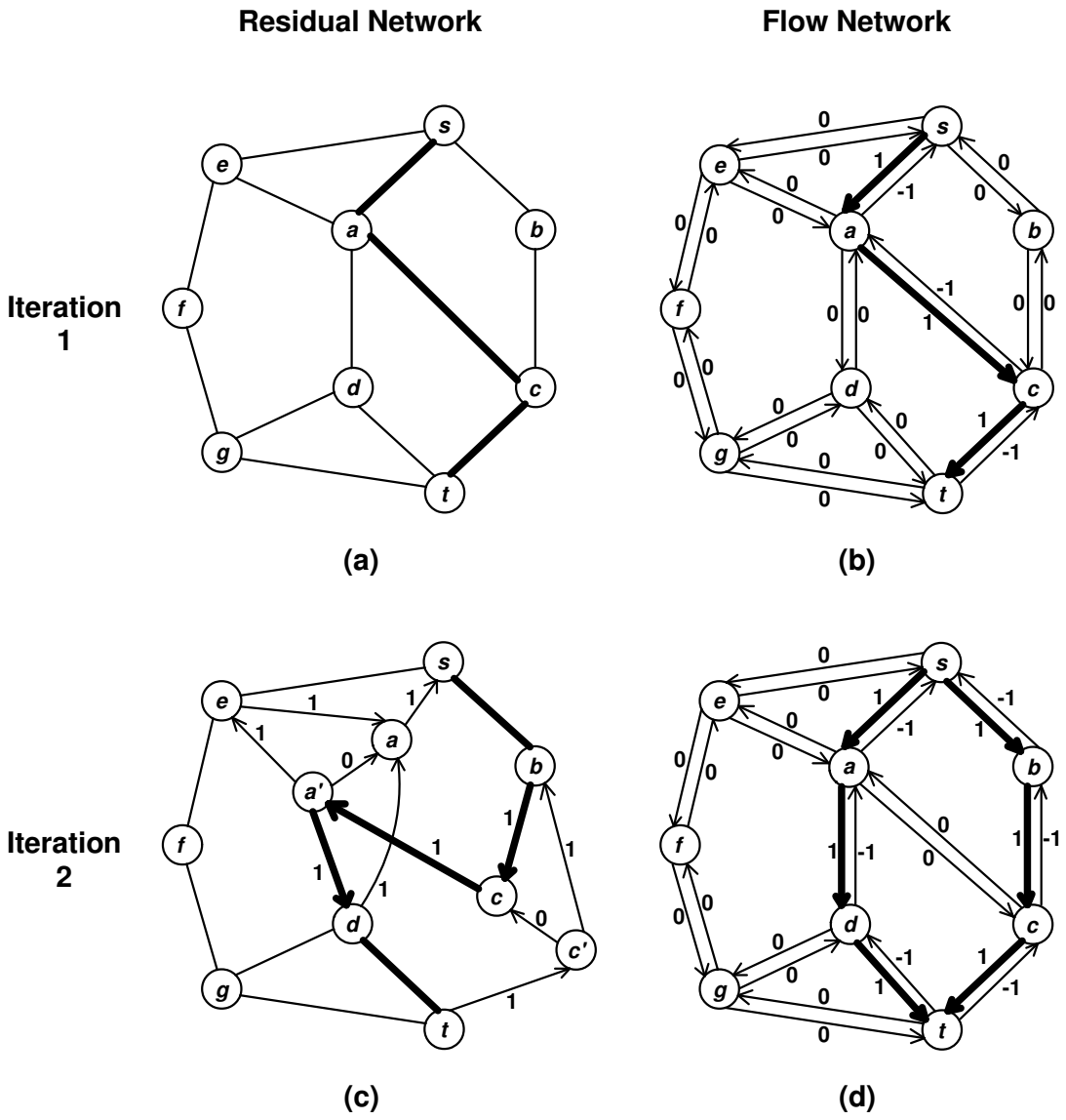


Figure 59: Two successive iterations of the execution of Counting-Paths for $k = 2$. (a) and (c) are the residual network G_{res} of each iteration with a bold augmenting path P from the source s to the sink t. (b) and (d) show the new *flow*. For clarity, the directed edges with zero capacity are not drawn in the residual network, except from the primed vertices to the original vertices.

1. **Split vertices**. This step explains line 3 in Algorithm 5. Each vertex on the shortest paths in the residual network G_{res}, except the source s and the sink t, is split into two vertices, namely the *original vertex* and the *primed vertex*. The two vertices are joined by a directed edge of zero capacity and directed from the primed vertex to the original vertex. This is illustrated in Figure 59(c). Vertices a and c are split into vertices a and a', c and c', respectively. We draw directed edges of zero capacity from a' to a and from c' to c. Details for other edges will be given in the following steps.

2. **Modify residual network**. This step explains line 4 in Algorithm 5. Recall that the residual network G_{res} is the network with residual capacity and the total capacity of each edge in our scenario is one. Therefore, for each edge (v, w) on the shortest paths, we have two cases:

   (a) If v is not the source vertex:
   $capacity_{\text{res}}(v', w) = capacity(v, w) - flow(v, w)$
   $capacity_{\text{res}}(w, v') = capacity(v, w) - capacity_{\text{res}}(v', w)$

   (b) If v is the source vertex:
   $capacity_{\text{res}}(v, w) = capacity(v, w) - flow(v, w)$
   $capacity_{\text{res}}(w, v) = capacity(v, w) - capacity_{\text{res}}(v, w)$

   In our example, $capacity_{\text{res}}(s, a)$, $capacity_{\text{res}}(a', c)$ and $capacity_{\text{res}}(c', t)$ in the residual network in Figure 59(c) are zero. However, for the clarity of the drawing purposes, the directed edges with zero capacity are not shown in the residual network's figure, except from the primed vertices to the original vertices. Moreover, $capacity_{\text{res}}(a, s)$, $capacity_{\text{res}}(c, a')$ and $capacity_{\text{res}}(t, c')$ are all one as shown in the figure.

3. **Replace external edges**. This step explains line 5 in Algorithm 5. We replace external edges connected to the vertices on the shortest paths with two oppositely directed edges of the same capacity, and connected to the two split-vertices. External directed edges terminate on the original vertices,

while they originate from the primed vertices. In the residual network in Figure 59(c), we need to replace external edges connecting to vertices a and c, i.e. (e, a), (d, a) and (b, c). Then, we draw directed edges of capacity one to the original vertices, i.e. from e to a, d to a, and b to c. We also draw the opposite directed edges from the primed vertices, i.e. from a' to e, a' to d, and c' to b. Note that other edges, i.e. edges in the residual network that are neither on the previously discovered shortest path nor incident to the vertices on the shortest path, are left unmodified.

4. **Find an augmenting path using a modified breadth first search**. This step explains how we find the shortest path from s to t in line 7 of the algorithm. In each iteration, we find an augmenting path from s to t that has the lowest path cost using the breadth first search technique. Recall that the path cost, denoted as $cost(P)$, is the total amount of cost to push each flow along each edge on the path in the residual network G_{res}. The cost is subtracted with the flow from the flow network if the direction of the path is opposite to the direction of the flow. We also add a little modification to breadth first search by giving advantage moves to the vertices on the previously discovered shortest paths, i.e. the split vertices. It means, when we discover a split vertex, we do not put it in the breadth first search's queue but examine it directly. This modification is aimed to tackle longer paths that are caused by overlapping edges. In our example network in Figure 59, there are two possible augmenting paths in the second iteration. They are $P_2 = \{s, b, c, a', d, t\}$ and $P_3 = \{s, e, f, g, t\}$. Breadth first search found that $cost(P_2) = 3$, because (c, a') has an opposite flow direction in Figure 59(b), while $cost(P_3) = 4$. Therefore, we take P_2 as the next augmenting path because it has the lowest path cost as shown in bold edges in Figure 59(c).

5. **Push flow**. This steps explains line 8 in Algorithm 5. If an augmenting path P exists, we merge the primed vertices with their original vertices. Then, the flow is pushed along P from s to t. Thus, for each edge (v, w) on P, we have:

$flow(v, w) \leftarrow flow(v, w) + 1$

$flow(w, v) \leftarrow -flow(v, w)$

Figure 59(d) shows the new flow after we push the flow along $P_2 = \{s, b, c, a, d, t\}$. Note that $flow(a, c)$ and $flow(c, a)$ are now zero.

6. **Remove overlapping edges**. This steps explains line 11 in Algorithm 5. We remove the overlapping edges of the paths found to obtain the shortest disjoint paths. The removal of overlapping edges can be done by crossing over the two paths. If we have two paths, say $P_1 = \{v_1, v_2, v_3, v_4\}$ and $P_2 = \{v_5, v_3, v_2, v_6\}$, the common edge is (v_2, v_3) or (v_3, v_2). When we cross over the two paths, the results are $P_1 = \{v_1, v_2, v_6\}$ and $P_2 = \{v_5, v_3, v_4\}$. In Figure 59, we have $P_1 = \{s, a, c, t\}$ and $P_2 = \{s, b, c, a, d, t\}$. These two paths share a common edge. i.e. (a, c) or (c, a). After removing the overlapping edge, the results are $P_1 = \{s, a, d, t\}$ and $P_2 = \{s, b, c, t\}$. The length of both paths is three.

We prove the correctness of the Counting-Paths algorithm by comparing to the Ford-Fulkerson algorithm and showing that in each of its iterations:

1. the modifications made to Ford-Fulkerson's residual graph, i.e. vertex-splitting and external edge replacement, do not remove any augmenting paths for breadth first search to find, and then, since breadth first search is complete, if there are currently $k-1$ paths in the collection of the disjoint paths, there must be at least one augmenting path remaining, and breadth first search must find this new path,

2. Counting-Paths always produces vertex-disjoint paths, and so once the size of the current collection of the disjoint paths is k, the algorithm terminates because it has found the maximum set of disjoint paths and there are no other augmenting paths remaining to be discovered by breadth first search.

We first show that both vertex-splitting and external edge replacement in the residual graph do not change the problem for breadth first search. A vertex v on the

previously discovered shortest path is split into two vertices v and v'. When v is split, the zero-length directed edge from v' to v enables breadth first search to include all possibilities of augmenting paths passing through v. Note that we call an edge that is not on the discovered shortest path but incident to v as an external edge. If the degree of v is two, v has no external edges because the two neighbours of v must also be on the discovered shortest path. If the degree of $v > 2$, v may be adjacent to one or more vertices that are not on the shortest path. Suppose there is a vertex w that is not on the shortest path and adjacent to v. In order for breadth first search to include all possibilities of augmenting paths from v to w and from w to v, the external edge is replaced with two directed edges from v' to w and from w to v, respectively, as illustrated in Figure 60. Since the vertex-splitting and the external edge replacement do not change things for the breadth first search part, if there are currently $k-1$ paths in the collection of the disjoint paths, breadth first search will find the last remaining augmenting path because it is complete.

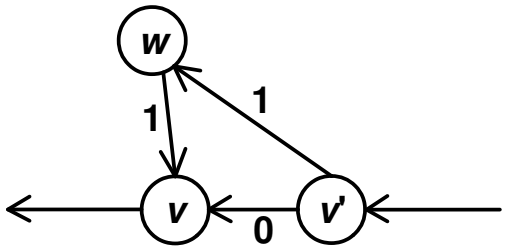


Figure 60: Vertex-splitting and external edge replacement in Counting-Paths

Secondly, we show that in each of its iterations, Counting-Paths produces disjoint paths. The Counting-Paths algorithm allows the edges of the new discovered shortest path to overlap with the previously found shortest paths. If there are some overlapping edges, Counting-Paths merges and reconstructs the paths by removing the overlapping edges that results in disjoint paths with no common edges and vertices, except the source and the destination vertices. Since at the end of each iteration it produces disjoint paths, it stops when the number of disjoint paths is k. When Counting-Paths terminates, k is the maximum set of disjoint paths and there are no augmenting paths from the source to the destination remaining.

Counting-Path is different to Modified Dijkstra [19] because it works with two graphs, i.e. residual and flow networks for Ford-Fulkerson's algorithm, while Modified Dijkstra only works with one graph. In Counting-Paths, edges on the shortest paths in the residual network are replaced with positive edges directed towards the source. In Modified Dijkstra, the directed edges have negative values.

5.2.2 Multiple Sources – Single Sink Problem

Multi-hop WSNs are often characterised by many-to-one (convergecast) traffic patterns, where many sources report data to a sink or a base station. In this kind of topology, we need to find whether all sources have k disjoint paths to the sink, so we need to use Counting-Paths repeatedly. If the network has n sources, we need to execute Counting-Paths n times which increases the worst time complexity to $O(|V||E|k)$. However, if we do not need to know the complete routing paths during the deployment process, it is not necessary for us to discover the actual paths, but only the number of disjoint paths and the neighbours that have k disjoint paths. This local information is used by nodes to forward their data to the nearest neighbours and the neighbours will decide where to forward them further. This motivates the dynamic programming variant of Counting-Paths, where we start counting the paths from sensor nodes closer to the sink.

In *finding k shortest disjoint paths for the multiple sources – single sink problem*, given a graph $G = (V, E)$, we check if a source $s \in V$ has k disjoint paths to a destination $t \in V$, $t \neq s$, by finding the k shortest disjoint paths, if they exist, from s to t or v, where $v \in V$ and v has k disjoint paths. Below, we prove the result that justifies our dynamic programming approach.

Lemma 5.1. *Let v be a vertex which has vertex-disjoint paths to a subset W of k vertices none of which have a cutset of size $< k$. Then v has no cutset of size $< k$.*

Proof. Suppose v does have a cutset, C, of size $< k$. A set of size $< k$ can break at most $k-1$ of the paths from v to W. Let $w \in W$ be any of the vertices whose

Algorithm 6: Counting-Paths-DP

Input : G, S, t, k
Output: $P_{i,j},\ \forall i = 1,\ldots,|S|,\ \forall j = 1,\ldots,k$

```
 1:  T ← {t}
 2:  for i ← 1 to |S| do
 3:      for j ← 1 to k do
 4:          if j > 1 then
 5:              Split vertices on the shortest paths except s_i ∈ S and r ∈ T
 6:              Modify the residual network G_res
 7:              Replace external edges connected to the vertices on the shortest paths
                 except s_i ∈ S and r ∈ T
 8:          end if
 9:          if there exists a path P_{i,j} from s_i ∈ S to r ∈ T in G_res then
10:              Push flow along P_{i,j} towards r
11:          end if
12:          if j > 1 then
13:              Remove overlapping edges
14:          end if
15:      end for
16:      if s_i ∈ S has k disjoint paths then
17:          T ← T ∪ {s_i}
18:      end if
19:  end for
20:  return P_{i,j}, ∀i = 1,...,|S|, ∀j = 1,...,k
```

paths from v are not broken by C, and so v is still connected to w. But w must be connected to S, since w has no cutset of size $< k$. Therefore v is still connected to S. Therefore C is not a cutset for v. Contradiction. □

As a corollary, if a vertex v has vertex-disjoint paths to k vertices, each of which has k vertex-disjoint paths to the sink, then v must also have k vertex-disjoint paths to the sink.[1]

In the dynamic programming variant of Counting-Paths, we start finding the k disjoint paths from vertices closer to the sink. For each vertex, if we can find k vertices that have k disjoint paths, we do not need to find the k disjoint paths to the sink and we can proceed to the next one. The algorithm for the dynamic programming variant of Counting-Paths is given in Algorithm 6. It takes as input

[1]We have not proven that the dynamic programming variant of Counting-Paths guarantees the length-bound, but in all problems that we tested in simulation it does obey the length-bound.

a graph G, a set S of source vertices, a destination t, and the number of disjoint paths sought k. T represents a collection of destination vertices, which are the sink and the vertices which have k disjoint paths to the sink. Note that line 3 to 15 are similar to Algorithm 5, but the shortest path may terminate at any vertices in T.

In the multiple sources – single sink problem, we vary the heuristic techniques to pick which vertex is examined first. Here are the heuristic techniques that we use:

1. **Breadth first search with the smallest vertex's ID to break ties**. We find the k disjoint paths from vertices which are closer to the sink first, so an m-hop vertex can use the information from its neighbours which are $(m-1)$-hop away from the sink. There must be several vertices with the same hop distances to the sink. In this case, we pick the one with the smallest ID.

2. **Breadth first search with the highest vertex's degree to break ties**. The same as the previous technique, but we choose the one with the highest degree if there are ties. If there still exists more than one vertex, we pick the one with the smallest ID.

3. **Breadth first search with the most processed neighbours to break ties**. Similar to the previous two, however, we select the one which has the most neighbours that have already known that they have k disjoint paths. If there are still ties, we choose the one with the smallest ID.

4. **Best first search with the most processed neighbours to break ties**. In this technique, we select a vertex which has the most neighbours with k disjoint paths. If there are ties, we take the one closer to the sink, i.e. the vertex with shorter hop count to the sink. But if there still exist several vertices that satisfy these two conditions, we choose the one with the smallest ID.

We will evaluate the contributions of these four heuristic techniques to the performance of the dynamic programming variant of Counting-Paths later in Section 5.3.2.

5.2.3 Single Source – Multiple Sinks and Multiple Sources – Multiple Sinks Problems

Two other variations of our problems are the single source – multiple sinks and multiple sources – multiple sinks problems. In these multiple sink cases, a well-known approach is by adding a *supersink* as an imaginary vertex that has connection to the original sinks. By doing this, we reduce the problem of single source – multiple sinks to the problem of single source – single sink, while the problem of multiple sources – multiple sinks is simplified to the problem of multiple sources – single sink.

When there are many sinks, we have two cases where the disjoint paths terminate at: *different-sinks* and *any-sinks*. The different-sinks problem is where the k disjoint paths must terminate at k different sinks to guarantee reliability of the network. Furthermore, the any-sinks problem is the case where the k disjoint paths may terminate at any sinks. In the different-sinks problem, for each connection from an original sink t to the supersink t', we set $capacity(t, t') = 1$, so the edge can be used at most once. However, for the any-sinks problem, we set $capacity(t, t') = k$, so the paths can traverse some original sinks more than once, but at most k times before reaching the supersink.

5.3 Evaluation of Counting-Paths

By the simulation, we want to show the efficiency and the accuracy of Counting-Paths compared to closely related algorithms. In the simulation, we use the following metrics to measure the performance of the algorithms:

1. ***Number of table lookups***. We want to evaluate the efficiency of Counting-Paths compared to other algorithms in terms of the total number of table access.

2. ***Runtime***. This metric also shows the efficiency of the algorithm. We want to compare the runtime of Counting-Paths to other algorithms. Because a disjoint path algorithm is used by a topology planning algorithm and is usually executed repeatedly for every node in a network, the shorter the runtime is the better.

3. ***Storage capacity***. This metric shows the efficiency of the algorithm to find disjoint paths by using the number of array cells needed to store graphs' information, which in turn becomes a good indication of the memory size required.

4. ***Number of disjoint paths***. We present this metric to show the accuracy of the algorithm. We expect that Counting-Paths can discover the maximum k such that a node has k disjoint paths and that the length difference between the shortest path and the longest path is minimal.

All algorithms are written in C++ and simulations are carried out in 2.40 GHz Intel Core2 Duo CPU with 4 GB of RAM. We do not use standard network simulators because we are not evaluating network protocols and operations, but rather the performance of algorithms that are used in planning a network. Our simulation results are based on the mean value of 20 different randomly generated network deployments. Note that we do not show error bars in graphs with logarithmic scale to improve readability of the graphs. The network consists of up to 100 nodes deployed within randomly perturbed grids of a two-dimensional area, where a node is placed in a unit grid square of 8 m $\times$ 8 m and the coordinates are perturbed. In this simulation, we generate 5 $\times$ 5, 7 $\times$ 7 and 10 $\times$ 10 grid squares to deploy 25, 49 and 100 nodes, respectively. Unless otherwise stated, we use 10 metres transmission range and assume that the communication graph follows the unit disk graph model and the links are bi-directional. We use the unit disk graph model for the ease of simulation purposes to have symmetrical links. However, any communication models can work with our algorithms as long as the links are symmetrical.

We compared the performance of Counting-Paths to the Modified Dijkstra algorithm by Bhandari [19] and the two algorithms proposed by Torrieri [113], namely Fast Pathfinding and Maximum Paths. These algorithms were reviewed in Section 2.5 and the pseudocode for each of them is given in Appendix B. We choose these algorithms because they have similar objectives to ours, i.e. finding k shortest disjoint paths between source nodes and sinks. We followed the three algorithms detailed in [19] and [113], implemented them and then verified the results by using the examples illustrated in the papers.

5.3.1 Single Source – Single Sink Problem

In each topology, the location of the sink is fixed at the top-left corner of the network and the location of the source is at the bottom-right corner, so as to maximise the distance between them. By the simulation of single source – single sink, we want to evaluate how many disjoint paths each algorithm can find, so we set $k = \infty$.

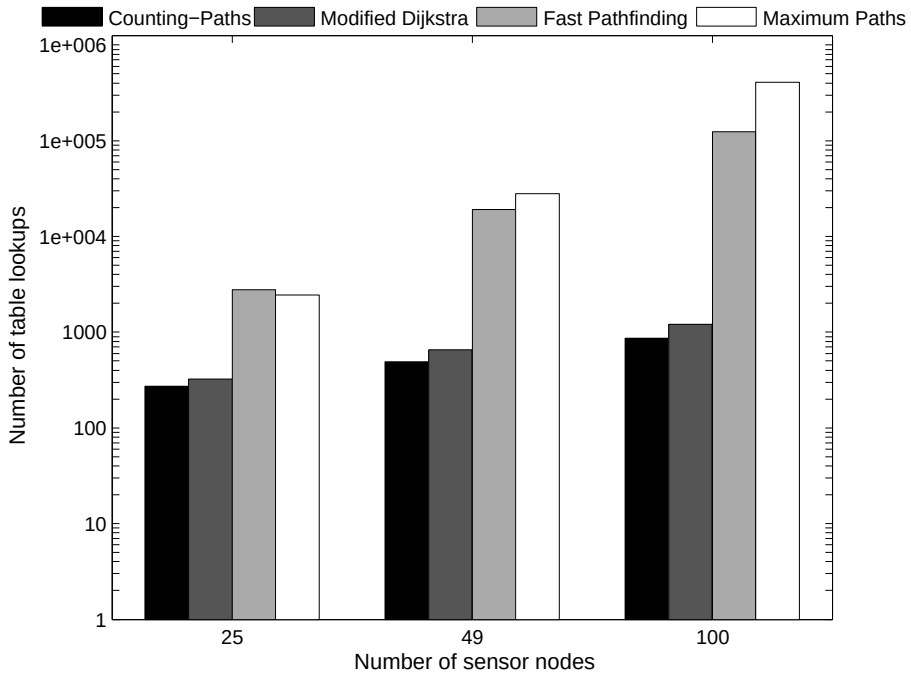


Figure 61: Number of table lookups versus number of sensor nodes for single source – single sink

Table 9: Disjoint paths algorithms' runtime for single source – single sink

Algorithms	Runtime (sec)		
	25-node	**49-node**	**100-node**
Counting-Paths	0.000313	0.001156	0.004609
Modified Dijkstra	0.000337	0.001212	0.004976
Fast Pathfinding	0.001140	0.002938	0.013624
Maximum Paths	8.001600	11.829600	16.473550

Figure 61 shows the total numbers of table lookups when the transmission range is 10 metres. The results show that Counting-Paths and Modified Dijkstra are more efficient than Fast Pathfinding and Maximum Paths. The numbers of table lookups for the two algorithm by Torrieri increase significantly when the number of nodes increases, because they try to find all possible combinations of paths. The more the sensor nodes, the more the combinations of paths discovered. The results also show that Counting-Paths has fewer table lookups compared to Modified Dijkstra, because in the worst case, breadth first search has lower complexity, i.e. $O(|V|+|E|)$, than Dijkstra's $O(|V|^2)$ [34]. These results correspond to the runtime in Table 9.

We want to show the impact of network density on the efficiency of the algorithms, in terms of the number of table lookups. Hence, we increase the transmission range of the sensor nodes to 15 metres and 20 metres to have denser networks. Even though the number of hops in the network becomes very low when we increase the transmission range, our objective here is only to compare the efficiency of the algorithms when the network density increases. We show the number of table lookups with increasing transmission range in 100-node networks in Figure 62. The numbers of table lookups for Counting-Paths and Modified Dijkstra slightly increase because when the network becomes denser, a sensor node has more neighbours to be visited by breadth first search and Dijkstra's algorithm. On the contrary, the numbers of table lookups for Torrieri's algorithms decrease. This happens because a sensor node has more neighbours in a dense network and when it is selected to be an intermediate node in a path, more neighbours are removed. This reduces the search space in the next iterations.

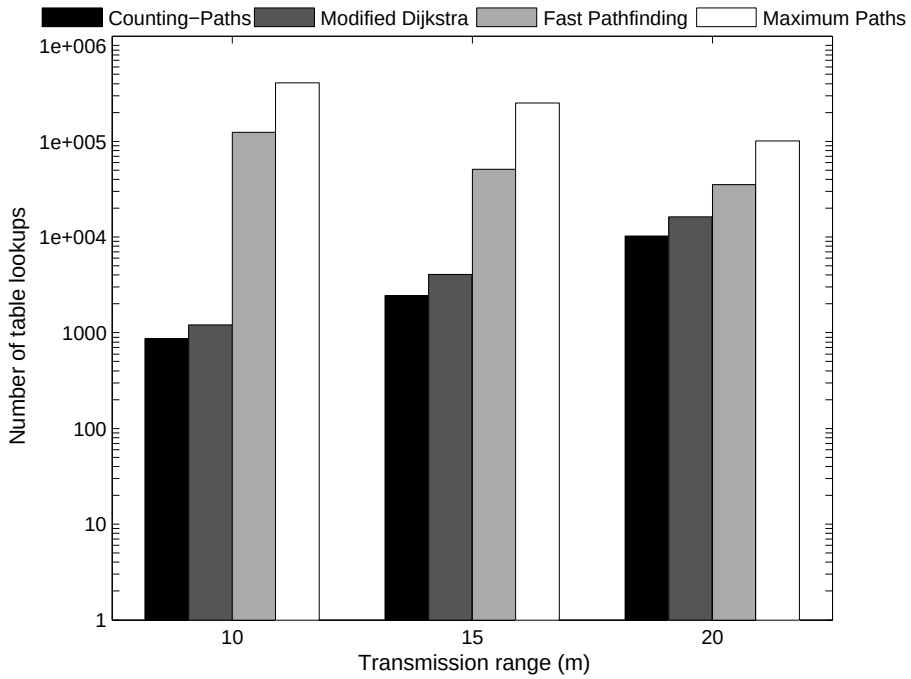


Figure 62: Number of table lookups versus transmission ranges for single source – single sink

We present the comparison of the storage capacities in Figure 63. Counting-Paths uses more storage than Modified Dijkstra because it has to maintain the residual network and the flow network for the Ford-Fulkerson algorithm, as well as a queue for breadth first search. Maximum Paths uses more storage than Fast Pathfinding because it stores all possible combinations of paths in each iteration, whereas Fast Pathfinding only stores sets of nodes to construct the paths.

Figure 64 depicts the average number of disjoint paths found. Counting-Paths and Modified Dijkstra discover more disjoint paths than Fast Pathfinding and Maximum Paths. This happens because the first two algorithms allow overlapping edges, which will then be removed, and paths reconstruction. However, in Fast Pathfinding and Maximum Paths, once a path is selected, the intermediate nodes are removed from further search.

The relationship between the average number of disjoint paths found and the path length in 100-node networks is presented in Figure 65. This figure shows that,

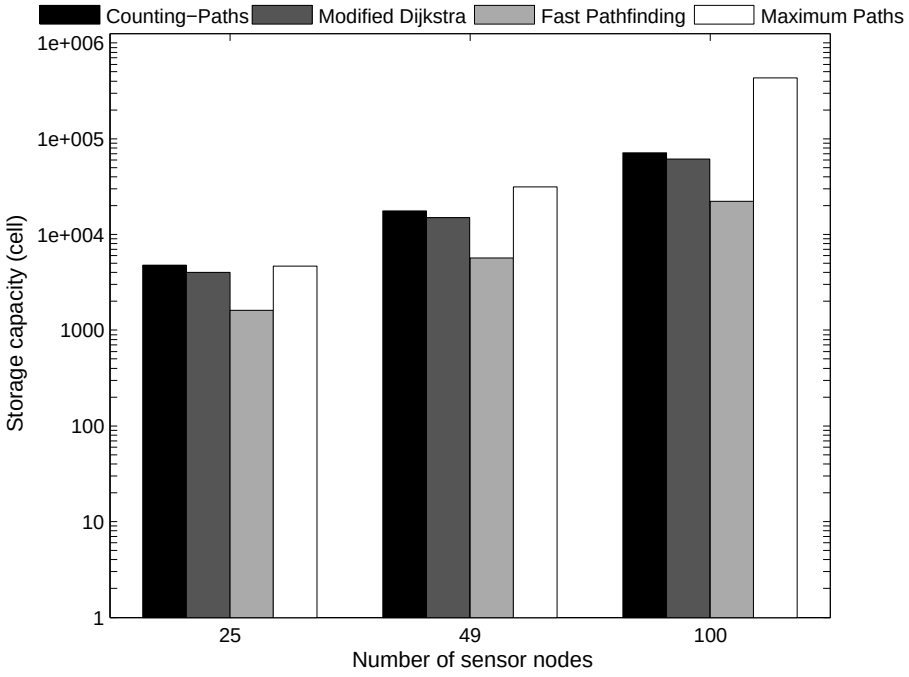


Figure 63: Storage capacity versus number of sensor nodes for single source – single sink

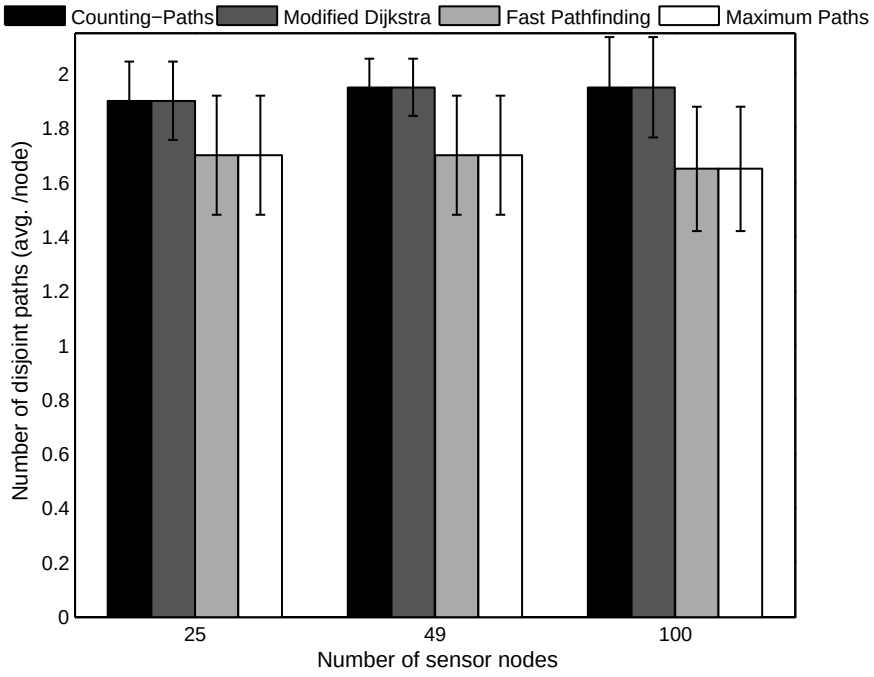


Figure 64: Number of disjoint paths versus number of sensor nodes for single source – single sink

on average, Counting-Paths and Modified Dijkstra find disjoint paths with lengths between 13 and 17 for the node at the bottom-right corner of the network and most of them are of length 15. Fast Pathfinding and Maximum Paths find longer paths, i.e. up to 18. These results are reasonable for our simulated topologies, where the average hop count from the source node to the sink is 15. We further compare Counting-Paths and Modified Dijkstra's results, especially for path lengths 14 to 16. The Modified Dijkstra algorithm discovers more shorter paths and more longer paths. On the other hand, Counting-Paths tries to balance the lengths of the paths found, where the length difference between the shortest path and other alternative paths is minimal.

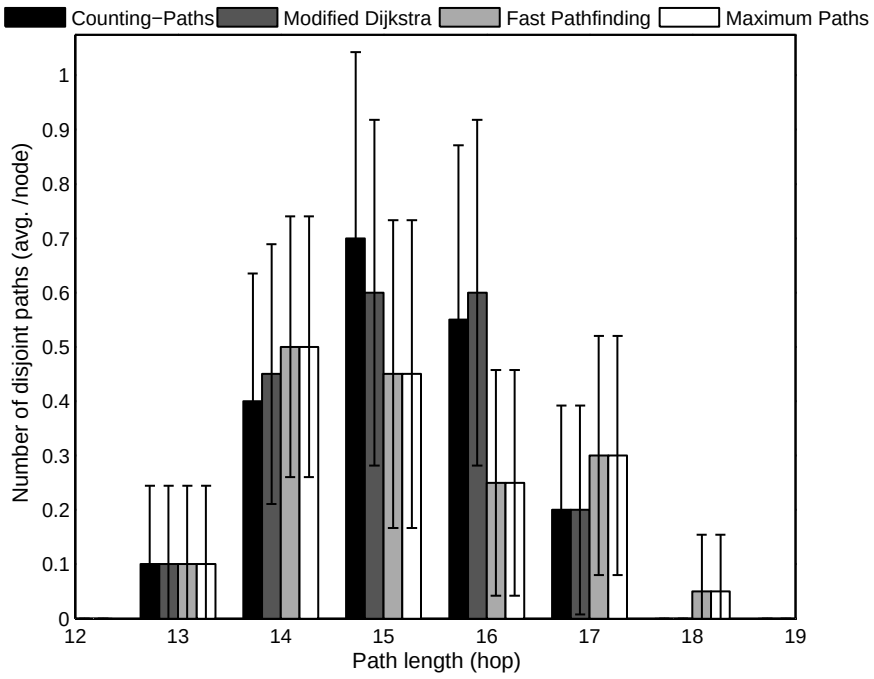


Figure 65: Number of disjoint paths versus path length for single source – single sink

5.3.2 Multiple Sources – Single Sink Problem

For the multiple sources – single sink problem, the location of the sink is still at the top-left corner of the network, while all sensor nodes are the source nodes.

In this simulation, we want to find whether all nodes in the network have two disjoint paths, so we set $k = 2$. We compare the performance of Counting-Paths to Modified Dijkstra, Fast Pathfinding and Maximum Paths. Besides using the basic Counting-Paths algorithm which is executed multiple times, we implement its dynamic programming (DP) variant and vary the heuristic techniques to select which node is examined first. In addition, we also implement Modified Dijkstra using the dynamic programming method.

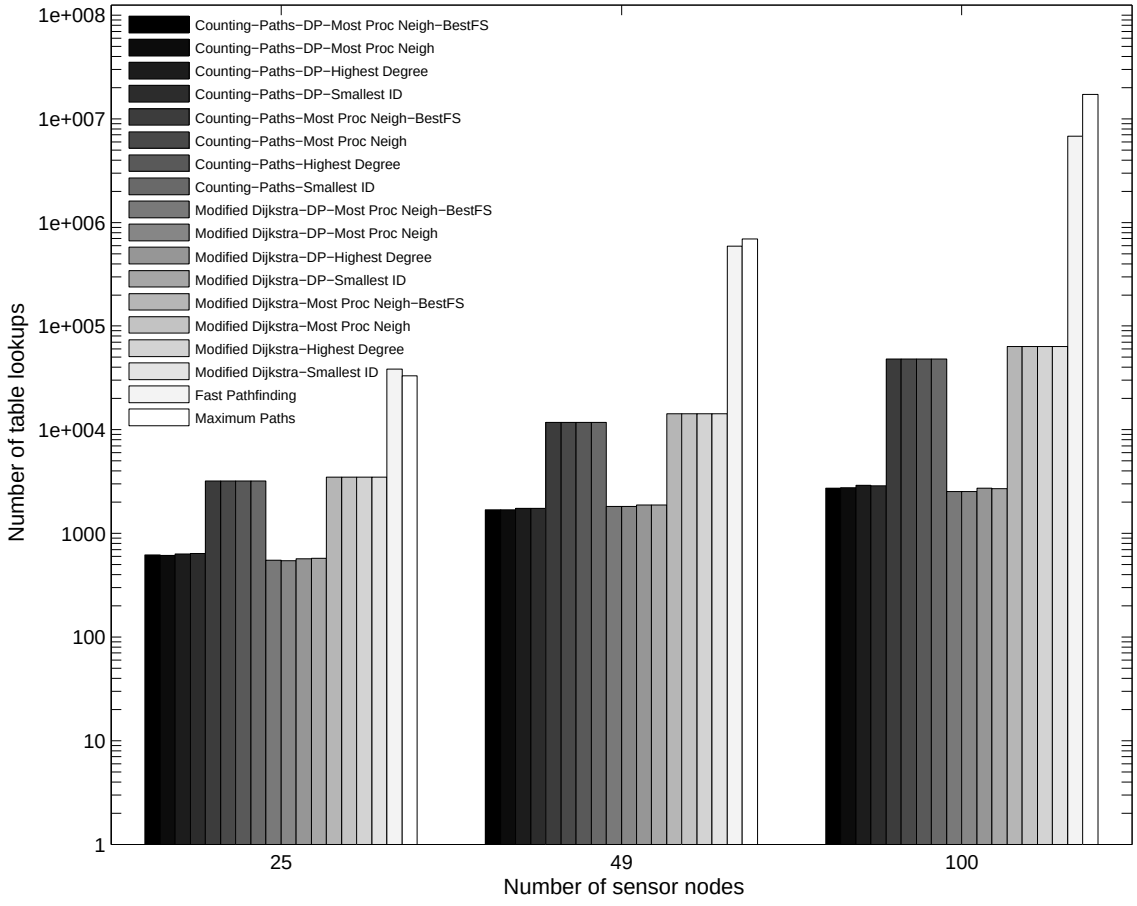


Figure 66: Number of table lookups versus number of sensor nodes for multiple sources – single sink

Figure 66 and Table 10 show the number of table lookups and the runtime of the algorithms, respectively, for the multiple sources – single sink problem. Without dynamic programming, the number of table lookups is not influenced by which node

Table 10: Disjoint paths algorithms' runtime for multiple sources – single sink

Algorithms	Runtime (sec)		
	25-node	**49-node**	**100-node**
Counting-Paths-DP-Most Proc Neigh-BestFS	0.000515	0.003626	0.027218
Counting-Paths-DP-Most Proc Neigh	0.000476	0.003299	0.023007
Counting-Paths-DP-Highest Degree	0.000588	0.003531	0.025343
Counting-Paths-DP-Smallest ID	0.000563	0.003485	0.025336
Counting-Paths-Most Proc Neigh-BestFS	0.001304	0.008187	0.062821
Counting-Paths-Most Proc Neigh	0.001351	0.008257	0.062984
Counting-Paths-Highest Degree	0.001328	0.008296	0.063125
Counting-Paths-Smallest ID	0.001258	0.008171	0.062727
Modified Dijkstra-DP-Most Proc Neigh-BestFS	0.000453	0.003055	0.022899
Modified Dijkstra-DP-Most Proc Neigh	0.000453	0.002953	0.022859
Modified Dijkstra-DP-Highest Degree	0.000492	0.003131	0.022913
Modified Dijkstra-DP-Smallest ID	0.000492	0.003102	0.022906
Modified Dijkstra-Most Proc Neigh-BestFS	0.000836	0.005751	0.047040
Modified Dijkstra-Most Proc Neigh	0.000851	0.005829	0.047104
Modified Dijkstra-Highest Degree	0.000826	0.005821	0.046295
Modified Dijkstra-Smallest ID	0.000851	0.005859	0.046985
Fast Pathfinding	0.015026	0.070181	0.577030
Maximum Paths	125.522800	373.737500	1010.415550

is selected first, because we have to find two disjoint paths from all nodes to the sink. In this case, Counting-Paths has the fewest number of table lookups compared to the other algorithms as has been shown in Figure 61 for the single source – single sink cases. However, the runtime of Counting-Paths is slightly longer than Modified Dijkstra, because Counting-Paths needs to repeatedly update two graphs, i.e. the residual network and the flow network, while Modified Dijkstra only works with one graph.

With dynamic programming, we show that we are able to reduce the complexity of the algorithms significantly when we only find k disjoint paths to k nearest neighbours that have already had k disjoint paths. In this case, Counting-Paths achieves around 5, 7, and 17.5 times improvement for the 25-node, 49-node, and 100-node topologies, respectively. Similarly, Modified Dijkstra also experiences improvements with the dynamic programming method. Moreover, comparing the heuristic techniques to pick the nodes, either breadth first search or best first search with the most processed neighbours has fewer number of table lookups compared to breadth

first search with the highest node's degree and the smallest node's ID. This happens because a node with more processed neighbours needs fewer shortest path iterations.

We show the storage capacity and the average number of disjoint paths found per node in Figure 67 and 68, respectively. Recall that in this simulation, we set $k=2$. We only present the results for Counting-Paths and Modified Dijkstra using one bar for each group because they have the same results regardless the variations on the experiments. We observe in these two figures that the storage capacity and the average number of disjoint paths found per node have similar trends with the single source – single sink cases, which have been shown in Figure 63 and 64

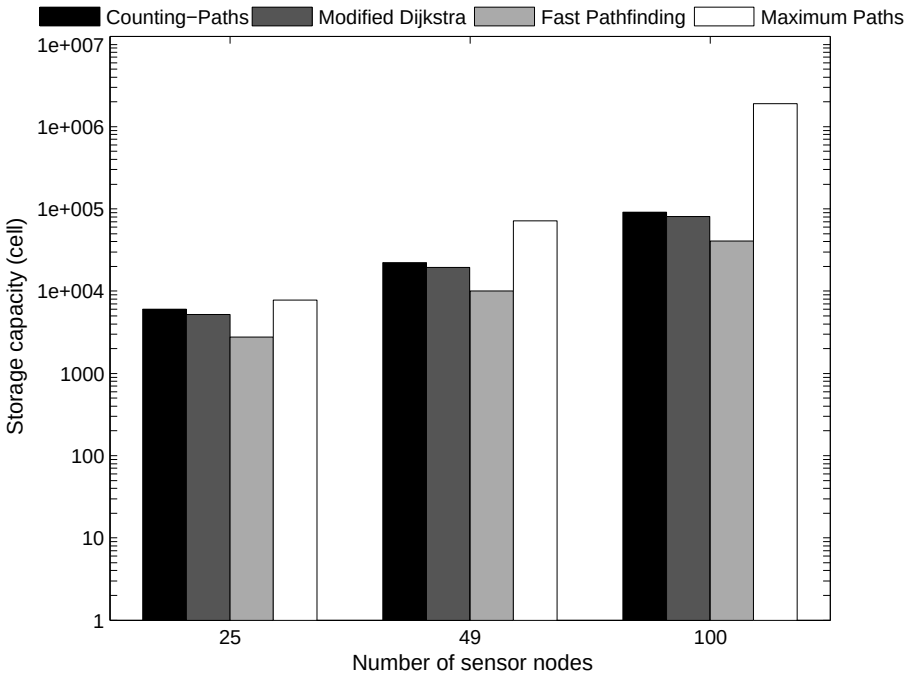


Figure 67: Storage capacity versus number of sensor nodes for multiple sources – single sink

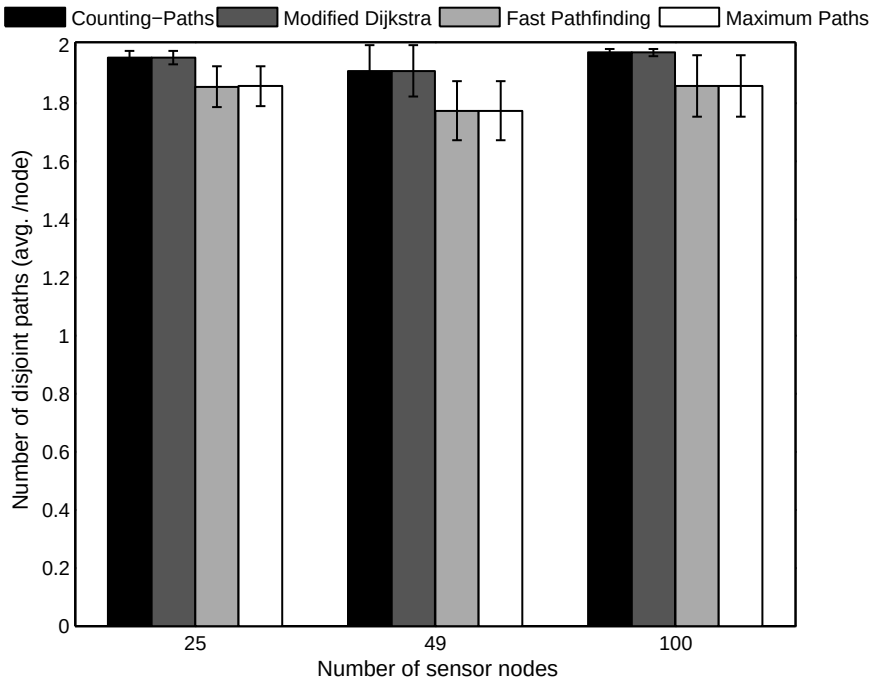


Figure 68: Number of disjoint paths versus number of sensor nodes for multiple sources – single sink

5.4 Greedy Randomised Adaptive Search Procedure for Additional Relay Placement (GRASP-ARP)

In this section, we introduce GRASP-ARP, a local search algorithm based on GRASP [41, 42, 96] to deploy additional relay nodes for ensuring the existence of k disjoint paths in WSNs with data sinks. GRASP-ARP requires repeated counting of the number of disjoint paths for each node, for which we use either Counting-Paths or its dynamic programming variant. With the basic Counting-Paths, GRASP-ARP ensures a length constraint. On the other hand, it has shorter runtime with the dynamic programming variant.

Recall that in this chapter, we focus our research on the single-tiered, constrained partial fault-tolerant relay placement problem. All of the algorithms reviewed in

Section 2.4 try to achieve k-connectivity, and are designed for WSNs without designated sinks. Unlike our algorithm, the published algorithms do not place any constraints on the lengths of the paths. On the other hand, our GRASP-ARP is designed to place a minimum number of relays in the existing network so that the sensors have k disjoint paths to the sinks, and to minimise the path length. The closest problem definition to ours is Misra *et al.*'s [81]. However, it can only establish up to 2-connectivity. Other work with similar objectives to ours include Bredin *et al.* [26], Pu *et al.* [90] and Han *et al.* [51], although they are for unconstrained deployment locations. Bredin *et al.*'s [26] considers full fault-tolerant relay node placement. This was then modified by Pu *et al.* [90] for partial fault-tolerance after noting that there is no need to ensure multiple paths for the relay nodes. The simulation results in Han *et al.*'s work [51] show that Bredin *et al.*'s [26] is more efficient for partial fault-tolerance, while Han *et al.*'s [51] is more efficient for full fault-tolerance in terms of the number of additional relay nodes. Therefore, among all existing algorithms that we review in this thesis, we infer that the most relevant one for our work is Pu *et al.*'s [90], except that it assumes unconstrained relay node locations, where relays are deployed along straight lines between pairs of sensor nodes. This assumption is unrealistic due to the existence of physical obstacles in the monitoring region. We make some modifications to Pu's algorithm, which is described in Section 5.5, to work in constrained deployment locations.

In [77, 78], Martins *et al.* use GRASP to solve the Steiner tree problem in graphs (SPG). SPG is similar to the relay placement problem, in that it must select from a set of candidate nodes in order to connect a number of designated terminals, although its aim is to find a minimal spanning tree rather than a forest with disjoint paths.

We consider WSNs where the vertices are partitioned into sensors, relays and sinks, and so in the graph representation $V = T \cup R \cup S$. We define a WSN to be *(k, l)-sink-connected* if and only if for every vertex $v \in T$, there are k disjoint paths from v to S of length $\leq l$.

We can now define the *additional relay placement problem*: given a graph $G = (T \cup A \cup S, E)$, where A is a set of candidate locations to deploy relays, find a minimal subset $R \subseteq A$ such that $H = (T \cup R \cup S, E\downarrow_{T \cup R \cup S})$ is (k, l)-sink-connected.

For our algorithm, we introduce some secondary definitions. k_v is the number of length-bounded disjoint paths a sensor currently has. $X \subseteq T$ is the set of sensors with $k_x < k, \forall x \in X$. $Y \subseteq T$ is the set of sensors with $k_y \geq k, \forall y \in Y$, and such that y is on a path of a sensor $x \in X$ to a sink or a vertex with at least k disjoint paths. $Z \subseteq T$ is a set of sensors with $k_z \geq k, \forall z \in Z$, such that z does not appear on a path of a sensor $x \in X$ to a sink or a vertex with k disjoint paths.

Before the execution of GRASP-ARP, we use Counting-Paths to find k disjoint paths from all sensors to sinks. Then, for each sensor, we count how many disjoint paths that satisfy the length restriction $l_{\max}$. Based on this result, we determine the sets X, Y, and Z, which become inputs to GRASP-ARP.

5.4.1 Construction Phase

The first step in any GRASP algorithm is to construct an initial solution. We generate a graph $G' = (V', E')$, where $V' = X \cup Z \cup S$ and $E' = \{(v, w), v \in X, w \in Z \cup S \mid \exists srp(v, w)\}$. $srp(v, w)$ denotes the shortest relay path from v to w in the original graph $G = (V, E)$ in terms of cost c, where all intermediate vertices in the path are candidate relays. Cost is a non-negative function $c : E \rightarrow \mathbb{R}$ associated with each edge. The edge's cost $c'(v, w) = srp(v, w)$ is associated with each edge $(v, w) \in E'$. After that, we find a minimum forest F of G' such that for each $v \in X$, we select $k - k_v$ least cost edges to $w \in Z \cup S$. Then, we replace the edges in F with the edges in the shortest relay paths in the original graph G and save this set of paths P. Two shortest relay paths from $v \in X$ to $w \in Z \cup S$ may overlap. Since we are dealing with vertex-disjoint paths, the selection of the second shortest relay path must exclude all candidate relays on the first shortest path.

To vary the local optima solutions found by the local search phase in each iteration, we vary the initial solution. In order to add randomisation to the initial solution, we make the following modification to the computation of the minimum forest F of G'. Instead of selecting edges with the least cost, we build a restricted candidate list with all edges $(v,w) \in E'$ such that $c'(v,w) \leq c'_{\min} + \alpha(c'_{\max} - c'_{\min})$, where $0 \leq \alpha \leq 1$. $c'_{\min}$ and $c'_{\max}$ denote the least and the largest costs among all unselected edges, respectively. Then, k–k_v edges are selected at random from the restricted candidate list. In the construction phase, the initial solution is pure greedy when $\alpha = 0$, while $\alpha = 1$ is equivalent to a random construction. Later in the simulation, we select the value of α randomly in each iteration.

5.4.2 Node-based Local Search

The next stage in a GRASP algorithm is to explore the neighbourhood of the initial solution, looking for lower cost solutions. Let R be the set of relays in the current forest F and $r_{\text{used}}(P)$ denotes the number of relays used in the current set of paths P. We explore the neighbourhood of the current solution by either adding a new relay $r \in A \setminus R$ into R, or by eliminating a relay $t \in R$ from R. In each iteration of the local search, the evaluation of elimination moves is performed only if there are no improving insertion moves.

5.4.3 Algorithm Description

We describe GRASP-ARP that consists of the construction phase and the node-based local search phase. The pseudocode for GRASP-ARP is given in Algorithm 7. The algorithm takes as input the original graph $G = (V, E)$, the set S of sinks, the set A of candidate relays, the set X of sensors with $k_x < k$, the set Z of sensors with $k_z \geq k$ but do not appear on any discovered paths, the number of disjoint paths sought k, and the number of iterations (*max_iterations*). Graph $G' = (V', E')$ is computed in line 2. The procedure is repeated *max_iterations* times. *max_iterations*

Algorithm 7: GRASP-ARP

Input : $G, S, A, X, Z, k, max\_iterations$
Output: R^*, P^*

1: $best\_value \leftarrow \infty$
2: Compute $G' = (V', E')$ and $c'(v, w)$, $\forall (v, w) \in E'$
3: **for** $i \leftarrow 1$ to $max\_iterations$ **do**
/* Construction phase */
4: Find F of $G' = (V', E')$ with R and P as the result
5: **do**
6: **do**
7: $insertion \leftarrow$ **false**, $elimination \leftarrow$ **false**
/* Insertion moves */
8: $best\_set \leftarrow R$, $best\_number \leftarrow r_{\text{used}}(P)$
9: **for** all $r \in A \setminus R$ **do**
10: Count disjoint paths $\forall x \in X$ from F^{+r}, result in P_{new}
11: **if** $r_{\text{used}}(P_{\text{new}}) < best\_number$ **then**
12: $best\_set \leftarrow R \cup \{r\}$, $best\_number \leftarrow r_{\text{used}}(P_{\text{new}})$
13: **end if**
14: **end for**
15: **if** $best\_number < r_{\text{used}}(P)$ **then**
16: $R \leftarrow R \cup \{r\}$, $F \leftarrow F^{+r}$, $P \leftarrow P_{\text{new}}$, $r_{\text{used}}(P) \leftarrow r_{\text{used}}(P_{\text{new}})$
17: $insertion \leftarrow$ **true**
18: **end if**
19: **while** *insertion*
/* Elimination moves */
20: $best\_set \leftarrow R$, $best\_number \leftarrow r_{\text{used}}(P)$
21: **for** all $t \in R$ **do**
22: Count disjoint paths $\forall x \in X$ from F^{-t}, result in P_{new}
23: **if** $r_{\text{used}}(P_{\text{new}}) < best\_number$ **then**
24: $best\_set \leftarrow R \setminus \{t\}$, $best\_number \leftarrow r_{\text{used}}(P_{\text{new}})$
25: **end if**
26: **end for**
27: **if** $best\_number < r_{\text{used}}(P)$ **then**
28: $R \leftarrow R \setminus \{t\}$, $F \leftarrow F^{-t}$, $P \leftarrow P_{\text{new}}$, $r_{\text{used}}(P) \leftarrow r_{\text{used}}(P_{\text{new}})$
29: $elimination \leftarrow$ **true**
30: **end if**
31: **while** *elimination*
/* Best solution update */
32: **if** $r_{\text{used}}(P) < best\_value$ **then**
33: $R^* \leftarrow R$, $P^* \leftarrow P$, $best\_value \leftarrow r_{\text{used}}(P)$
34: **end if**
35: **end for**
36: **return** R^*, P^*

determines the stopping criterion of a GRASP algorithm. The larger the number of iterations, the larger will be the computation time and the better will be the solution found. In each iteration, a greedy randomised solution for a minimum forest F of G' is constructed in line 4. Let R be the set of relays in the current forest F, P be the set of paths, and $r_{\text{used}}(P)$ be the number of relays used in P.

The local search starts with the initialisation of the best set and the best number of relays in line 8. The best set of relays is the set of relays in the current best solution, while the best number of relays is the number of relays used in the set of disjoint paths. The loop from line 9 to 14 searches for the best insertion move. In line 10, we count the number of disjoint paths, which we can use either the basic Counting-Paths algorithm or its dynamic programming variant, from each sensor $x \in X$, defined by the insertion of vertex r into the current set of relays. Let P_{new} be its new set of paths. In line 11, we check if this new solution P_{new} improves the number of the relays used in the current best solution. The best set and the best number of relays are updated in line 12. When all insertion moves have been evaluated, we check in line 15 if an improving solution has been found. If the insertion moves produce a better solution, the set of relays, the minimum forest, the set of paths and the number of relays used are updated in line 16, and the local search continues.

If no improving solution is found from the insertion moves, the elimination moves from line 21 to 26 are evaluated. We reinitialise the best set and the best number of relays in line 20. We count in line 22 the number of disjoint paths from each sensor $x \in X$, defined by the elimination of v the current set of relays. In line 23, we check if this new solution P_{new} improves the number of the relays used in the current best solution. Then, the best set and the best number of relays are updated in line 24. Once all elimination moves have been evaluated, we check in line 27 if an improving solution has been found. If the elimination moves produce a better solution, the set of relays, the minimum forest, the set of paths and the number of relays used are updated in line 28, and the local search continues.

If, at the end of the local search, we found a better solution compared to the best solution found so far, we update in line 33 the set of relays, the set of paths and the least number of relays used. The best set R^* of relays and the best set P^* of paths are returned in line 36.

5.4.4 Acceleration Scheme

We follow the acceleration scheme for the node-based local search as in [77] by using faster implementation of the insertion moves (line 8–18). The idea is to keep a candidate list with promising insertion moves, which is periodically updated. The results in [77] show that this technique can significantly reduce the computational times. Even though there is a risk that a better solution is missed, Martins *et al.* have shown that the losses in terms of solution quality are very small.

The candidate list containing the *k_best* improving insertion moves is generated in the first iteration. The list is kept in an increasing order of the associated move values. In each following iteration, instead of reevaluating all insertion moves, we only evaluate moves from vertices in the candidate list. Each time a vertex is evaluated, it is removed from the list. When the list becomes empty, a new full iteration is performed, all insertion moves are evaluated, and the candidate list is rebuilt.

5.5 Evaluation of GRASP-ARP

In the simulation, we want to show the effectiveness and efficiency of GRASP-ARP compared to closely related algorithms. We use the following metrics to measure the performance of the algorithms:

1. ***Number of additional relay nodes***. This metric shows the effectiveness of the algorithm by the total number of deployed relay nodes in the network.

2. ***Runtime***. This metric measures the efficiency of the algorithm. We want to compare the running time of GRASP-ARP to other algorithms. Even though this is an offline process, shorter runtime is important especially when a new topology is evaluated as a result of either moving or adding/removing a node.

The sensor nodes are deployed in randomly perturbed grids, where a sensor node is placed in a unit grid square of 8 m $\times$ 8 m and the coordinates are perturbed. In this simulation, we want the original topologies as sparse as possible, because sufficiently dense networks do not need additional relays to guarantee the existence of disjoint paths. For example, most nodes in the topologies used in the experiments in Section 5.3 have two disjoint paths (see Figure 68). In order to get sparse networks (average degree 2-3), we generate more grid points than the number of nodes. For example, we use 6 $\times$ 6, 8 $\times$ 8 and 11 $\times$ 11 grid squares to randomly deploy 25, 49 and 100 nodes, respectively. This setup generates sparser topologies than those used in Section 5.3. Candidate relays are also distributed in a grid area, where a candidate occupies a unit grid square of 6m $\times$ 6m. For 25-node, 49-node and 100-node topologies, we use 49, 100 and 196 candidate relays, respectively. Both sensor and relay nodes use the same transmission range, i.e. 10 metres. The maximum path length ($l_{\max}$) is set to 10 for 25-node, 15 for 49-node and 20 for 100-node networks.

We compared the performance of GRASP-ARP to Partial k-Connectivity-Repair proposed by Pu *et al.* [90]. We choose this algorithm because it has similar fault-tolerant requirements with ours, i.e. partial fault-tolerance. k-Connectivity-Repair was originally proposed by Bredin *et al.* [26] for full fault-tolerant relay placement, which was reviewed in Section 2.4. It was then modified by Pu *et al.* [90] for partial fault-tolerance. We followed the Partial k-Connectivity-Repair algorithm detailed in [90] and made two necessary modifications to work in constrained deployment locations, where a relay can only be placed at a specific candidate location. The two modifications are as follows and the detailed pseudocode is given in Appendix C.

1. The original Partial k-Connectivity-Repair algorithm deploys relay nodes along a straight line between two sensor nodes. Therefore, our first modification is to place relay nodes in candidate locations along the shortest relay path between two sensor nodes.

2. When all relay nodes are deployed, we add our second modification by trying to remove relay nodes one by one by still preserving the node k-connectivity. The connectivity is checked using a maximum network-flow-based checking algorithm [86] as is used in [95].

5.5.1 Multiple Sources – Single Sink Problem

We compared GRASP-ARP against the Partial k-Connectivity-Repair algorithm (K-CONN-REPAIR for short) in the multiple sources – single sink scenario. In this simulation, we put the sink at the top-left corner or in the centre of the network, while all sensor nodes are the source nodes. In this simulation, we want to create networks with 2-connectivity and 3-connectivity, so we use $k=2$ and $k=3$. We simulate GRASP-ARP using the dynamic programming (DP) variant and the basic Counting-Paths algorithm to find the k disjoint paths. For the dynamic programming variant, we use the best first search with the most processed neighbours as the heuristic technique to pick which sensor node is examined first, because it has been shown as one of the best heuristics in the evaluation of Counting-Paths section. For the basic Counting-Paths algorithm, we use breadth first search with the smallest node's ID to select nodes, because the performance of the algorithm is not influenced by which node is selected first.

Figure 69 shows the number of additional relay nodes needed for $k=2$ and $k=3$ for the case where the sink location is at the top-left corner of 25-node networks, while Figure 70 shows the results for the case where the sink is in the centre of the networks. GRASP-ARP finds nearly the same number of additional relay nodes compared to K-CONN-REPAIR when the position of the sink is in the corner of the

network and $k=3$. However, it needs fewer relay nodes for $k=2$ because K-CONN-REPAIR deploys excessive relays for k-connectivity for each pair of nodes. When the position of the sink is in the centre of the network, GRASP-ARP outperforms K-CONN-REPAIR in terms of the number of additional relays for both $k=2$ and $k=3$ because the sink has higher connectivity in the centre of the network. These two figures also show that higher number of maximum iteration in the local search produces better results.

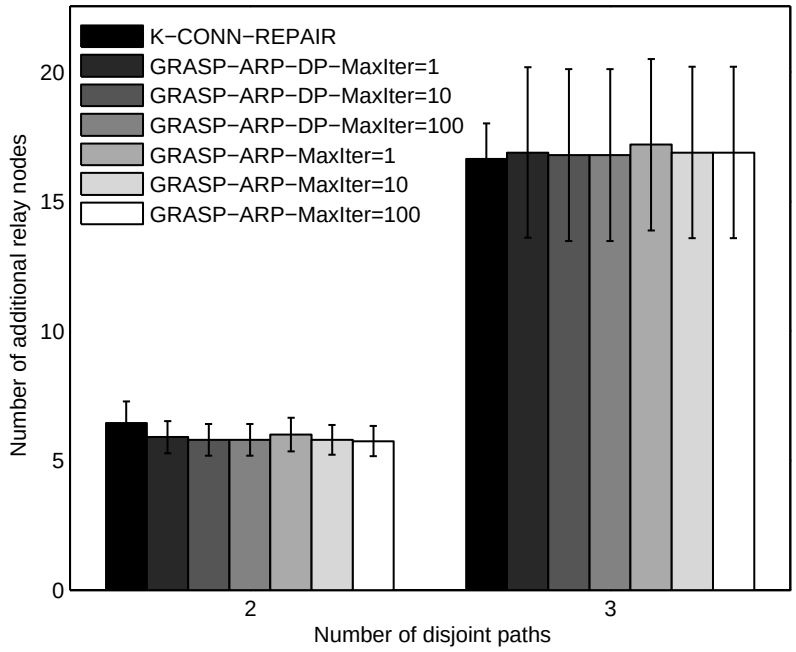


Figure 69: Number of additional relay nodes needed versus number of disjoint paths required for multiple sources – single corner sink in 25-node networks

Table 11 shows the algorithms' runtime, where for small networks, GRASP-ARP takes longer compared to K-CONN-REPAIR especially for $k=3$. This happens because GRASP-ARP repeatedly executes the Counting-Paths algorithm during the local search phase to find disjoint paths. GRASP-ARP with the dynamic programming variant of Counting-Paths (GRASP-ARP-DP) is faster than with the basic Counting-Paths (GRASP-ARP) because the dynamic programming has lower complexity.

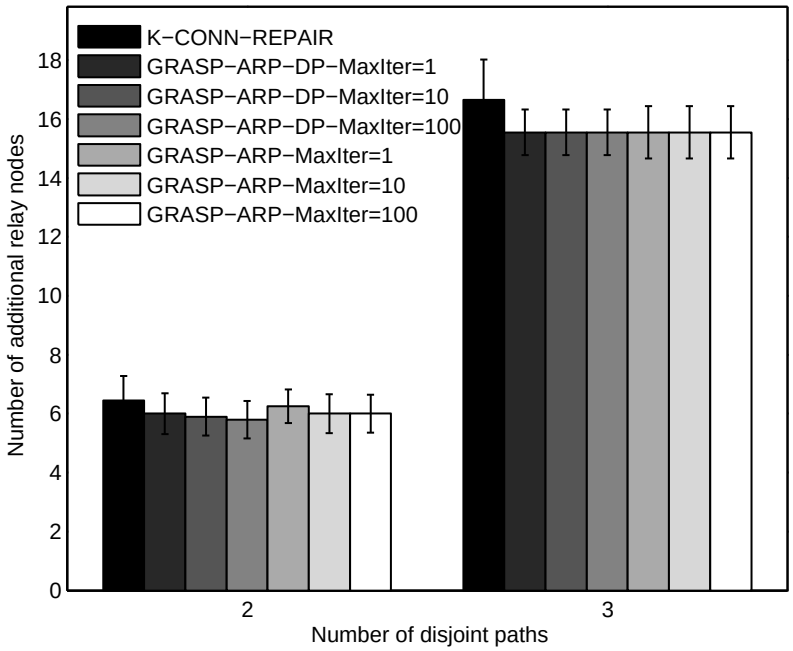


Figure 70: Number of additional relay nodes needed versus number of disjoint paths required for multiple sources – single centre sink in 25-node networks

Table 11: Additional relay placement algorithms' runtime for multiple sources – single sink in 25-node networks

Algorithms	**Runtime (sec)**	
	$k=2$	$k=3$
K-CONN-REPAIR	6.2891	7.4930
Sink at the corner		
GRASP-ARP-DP-MaxIter=1	5.6796	11.0656
GRASP-ARP-DP-MaxIter=10	24.3469	36.8148
GRASP-ARP-DP-MaxIter=100	185.0354	291.5102
GRASP-ARP-MaxIter=1	9.7672	19.2712
GRASP-ARP-MaxIter=10	39.4860	59.1335
GRASP-ARP-MaxIter=100	331.5993	459.0313
Sink at the centre		
GRASP-ARP-DP-MaxIter=1	3.4163	8.5166
GRASP-ARP-DP-MaxIter=10	9.7031	20.8140
GRASP-ARP-DP-MaxIter=100	73.1475	144.7258
GRASP-ARP-MaxIter=1	5.6696	14.2375
GRASP-ARP-MaxIter=10	16.7616	34.3545
GRASP-ARP-MaxIter=100	129.4321	239.1384

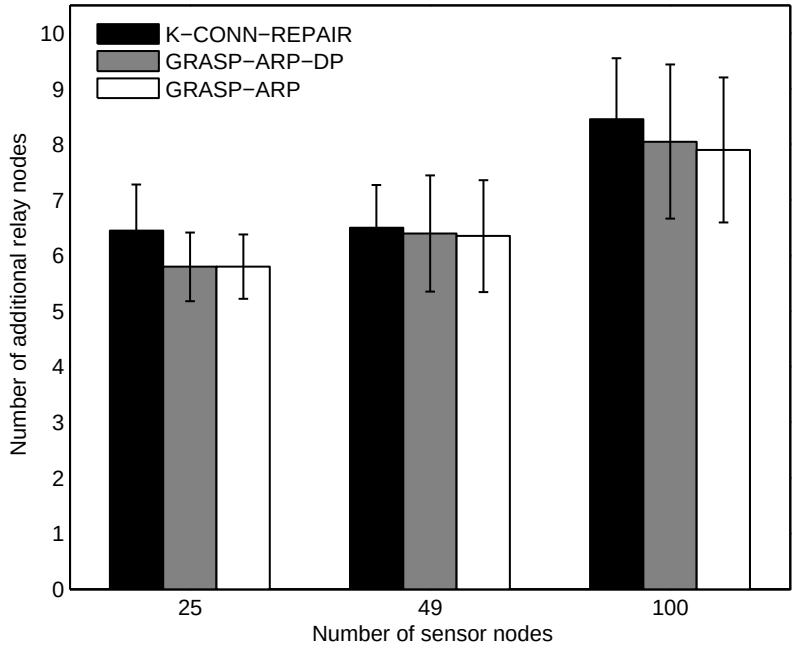


Figure 71: Number of additional relay nodes needed versus number of sensor nodes for multiple sources – single corner sink

Table 12: Additional relay placement algorithms' runtime for multiple sources – single sink

Algorithms	**Runtime (sec)**		
	25-node	**49-node**	**100-node**
K-CONN-REPAIR	6.2891	254.7343	10,003.8000
GRASP-ARP-DP	24.3469	421.2829	7,897.9650
GRASP-ARP	39.4860	619.3118	13,964.9400

We then extend our simulation to larger networks up to 100 nodes. Figure 71 depicts the number of additional relay nodes needed for 25, 49 and 100-node networks. In this simulation, we use $k = 2$ and set the sink position at the top-left corner of the network. We use *max_iteration* = 10 for GRASP-ARP because this number of iteration produces similar results to 100 iterations as shown in Figure 69 and 70. In all sizes of networks, GRASP-ARP outperforms K-CONN-REPAIR with fewer additional relay nodes. The algorithms' runtime is presented in Table 12, where we show that GRASP-ARP-DP is faster for bigger problems, i.e. 100-node networks, and so it scales better.

5.5.2 Multiple Sources – Multiple Sinks Problem

We compare GRASP-ARP against K-CONN-REPAIR in the multiple sources – multiple sinks scenario. We use the same simulation settings as in the previous multiple sources – single sink cases. However, in this simulation, we have four sinks deployed at the top-left, top-right, bottom-left and bottom-right corners of the network, while all sensor nodes are the source nodes. We simulate GRASP-ARP using the dynamic programming variant and the basic Counting-Paths algorithm. Recall that in the multiple sink problem, there are two cases where the disjoint paths terminate at: *different-sinks* and *any-sinks*. The different-sinks problem is where the k disjoint paths must terminate at k different sinks, while the any-sinks problem is the case where the k disjoint paths may terminate at any sinks. We will consider these two cases in the simulation.

Figure 72 shows the number of relay nodes required for $k=2$ and $k=3$ in 25-node networks, while Table 13 shows the runtime of the algorithms. GRASP-ARP results shown here are the simulation results with *max_iteration* = 10. The results show that GRASP-ARP in the multiple sources – multiple sinks scenario outperforms K-CONN-REPAIR with at least 50% fewer additional relays. This happens because GRASP-ARP only finds k disjoint paths to the dedicated sinks, either different sinks or any sinks. On the other hand, K-CONN-REPAIR must run an expensive

Table 13: Additional relay placement algorithms' runtime for multiple sources – multiple sinks in 25-node networks

Algorithms	**Runtime (sec)**	
	$k=2$	$k=3$
K-CONN-REPAIR	6.2891	7.4930
GRASP-ARP-AnySinks-DP	1.7844	10.9984
GRASP-ARP-AnySinks	2.5180	16.5088
GRASP-ARP-DiffSinks-DP	1.6351	11.0267
GRASP-ARP-DiffSinks	2.3828	18.8875

connectivity checking algorithm in each of its iterations to provide k-connectivity for an entire network. The results also show that the different-sinks case requires more relays than the any-sinks case because disjoint paths must be established to different sinks.

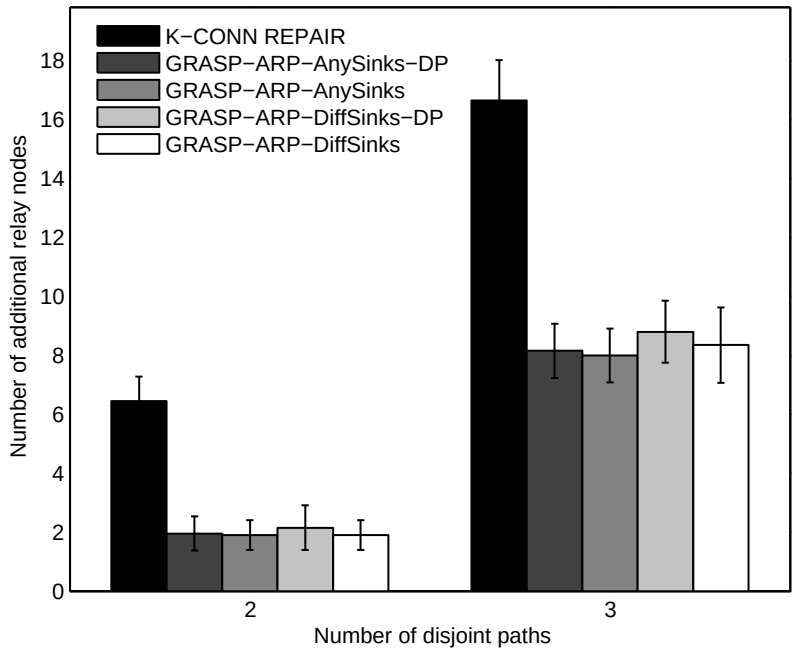


Figure 72: Number of additional relay nodes needed versus number of disjoint paths required for multiple sources – multiple sinks in 25-node networks

Figure 73 and Table 14 show the simulation results when we extend the simulations to larger networks. In this simulation, we use $k = 2$ and *max_iteration* =

Table 14: Additional relay placement algorithms' runtime for multiple sources – multiple sinks

Algorithms	**Runtime (sec)**		
	25-node	**49-node**	**100-node**
K-CONN-REPAIR	6.2891	254.7343	10,003.8000
GRASP-ARP-AnySinks-DP	1.7844	37.7884	517.3695
GRASP-ARP-AnySinks	2.5180	55.3843	735.7915
GRASP-ARP-DiffSinks-DP	1.6351	31.3648	426.6710
GRASP-ARP-DiffSinks	2.3828	51.4437	822.0718

10 for GRASP-ARP. In all network's sizes, GRASP-ARP outperforms K-CONN-REPAIR with fewer additional relay nodes and faster runtime. For 100-node networks, GRASP-ARP deploys 35% fewer relays with 10–20 times faster than K-CONN-REPAIR.

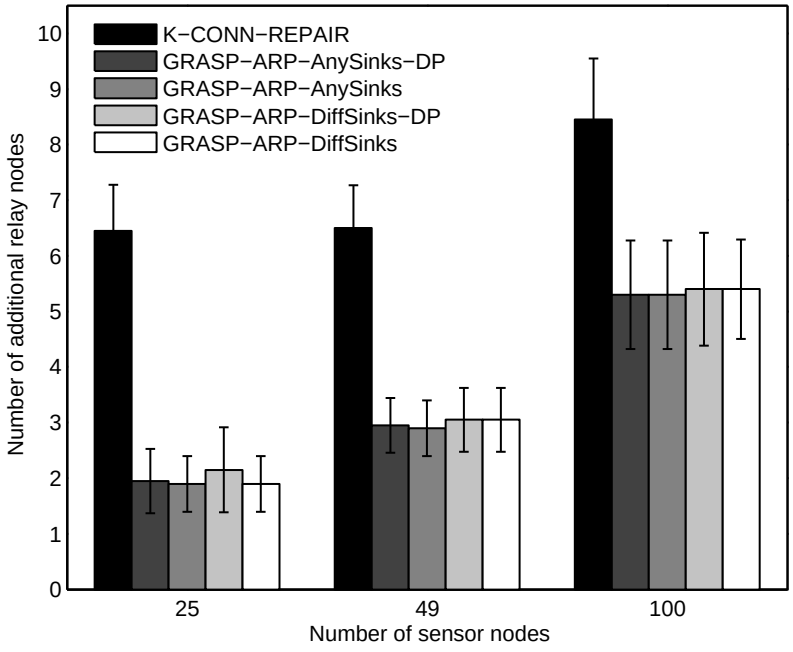


Figure 73: Number of additional relay nodes needed versus number of sensor nodes for multiple sources – multiple sinks

5.6 Conclusion

Ensuring that WSNs are robust to failures requires that the physical network topology will offer alternative routes to the sink. This requires a sensor network deployment to be planned with an objective of ensuring some measure of robustness in the topology, so that when failures do occur that routing protocols can continue to offer reliable delivery. In this chapter, we propose solutions for fault-tolerant WSN deployment planning by judicious use of additional relay nodes. We define the problem for increasing WSN reliability by deploying a number of additional relay nodes to ensure that each sensor node in the initial design has k disjoint paths with a length constraint to the sinks. We present two offline algorithms to be run during the initial topology planning to solve this problem. Counting-Paths uses the Ford-Fulkerson maximum flow algorithm to count the number of disjoint paths from each sensor node to the sinks and to find the k shortest disjoint paths. GRASP-ARP is a local search algorithm that modifies the existing GRASP algorithm to deploy a minimum number of additional relay nodes at the possible candidate locations. We also adapt a version of the closest approach from the literature for comparison. Our simulation results show that our solution requires fewer relay nodes for larger problems than the competitor, and that different variants of our algorithm are significantly faster, allowing us to tackle larger problems.

Chapter 6

Fault-Tolerant Relay Deployment Based on Length-Constrained Connectivity and Rerouting Centrality

6.1 Introduction

In this chapter, we propose a novel solution to reduce the deployment cost and the computation time of relay node placement in Wireless Sensor Networks (WSNs) with sinks. Our solution places a minimum number of relays as backup nodes to provide length-bounded alternative paths only around the most important (or critical) nodes. We present a new centrality metric to measure each node's importance in a network and then use a local search to minimise the number of relays that need to be deployed.

The standard approach to robustness is to provide k-connectivity [26, 51, 19] – that is, to ensure that the network will remain connected after the failure of any $k-1$ nodes. However, achieving k-connectivity requires an excessive number of relays,

and so some researchers proposed partial k-connectivity [60, 20], where not all nodes have k disjoint paths. Another approach is to consider the relative importance of each node for delivering data to the sink from other nodes. If the failure of a node would disconnect many other nodes, or cause traffic from many other nodes to be delivered late, then the node is important, and we should ensure alternative paths around that node. The importance of a node in network analysis is called its *centrality* [44, 25], and we introduce definitions of centrality which measure a node's impact on connectivity and path length for the rest of the network. We use this centrality measure as a priority order for providing alternative paths. If we have limited resources, we address only nodes with high centrality, with the intention of being robust to the most significant failures; in cases where we have more resources, we can address nodes with lower centrality, and provide robustness against more failures.

Specifically, we define *Length-constrained Connectivity and Rerouting Centrality* (l-CRC), a new centrality index for WSNs with sinks. This centrality index is a pair of values. The first value measures the importance with respect to network connectivity under a path length constraint, while the second value measures the additional length of shortest paths that would be required after a node fails. We introduce the single-tiered constrained fault-tolerant additional backup placement problem (ABP), in which we must find a minimal subset from a limited set of possible candidate positions to deploy relays, so that when a sensor node in a WSN dies, each of its descendants has an alternative length-constrained path to a sink. We use the centrality index to determine the most critical nodes, and to assess the quality of positions for the relays to provide alternative paths around the nodes with high centrality. To decide whether a node is critical or not, we use a threshold. A node is critical if its centrality index is above the threshold. We can raise the threshold to trade-off deployment cost for robustness. We introduce *Greedy Randomised Adaptive Search Procedure for Additional Backup Placement* (GRASP-ABP), a local search algorithm based on GRASP [41, 42, 96], which searches for the smallest

number of additional relays to ensure all sensor nodes have centrality measures below the threshold. We run GRASP-ABP as a centralised offline algorithm during the initial topology planning stage.

The rest of this chapter is organised as follows. We present l-CRC in Section 6.2 and GRASP-ABP in Section 6.3. We show our simulation results in Section 6.4. We compare GRASP-ABP against the closest approaches from the literature, and we demonstrate empirically that it produces networks with fewer additional relays, and scales effectively, requiring shorter runtime than the competitors for problems with hundreds of nodes. As we raise the threshold on the centrality indices, nodes that are identified as critical decrease, and so both the runtime and the number of required relay nodes drop significantly.

6.2 Length-constrained Connectivity and Rerouting Centrality (l-CRC)

The usual centrality indices measure the importance of a node based on the existence of the node in a network. For example by calculating how likely it falls on the shortest paths of other nodes (betweenness centrality) or how many neighbours it has (degree centrality). For dealing with failures, we cannot use these centrality indices, because we need one that defines the importance of a node based on the effect of removing the node from the network. The ones proposed by Shavitt and Singer [101, 102] calculate the indices based on the existence of a backup path if a node fails. However, they are designed for mesh networks, where the backup paths are sought between every pair of nodes. In WSNs, we only require backup paths between nodes and sinks. In addition, these centrality indices do not take into account the path length constraints. Since all existing centrality indices cannot be used to measure node criticality in the context of WSNs with sinks, where a length constraint is important, a new centrality index is needed.

Given the importance of connectivity and latency requirements in designing a reliable WSN topology, we define a new variation of alternative path centrality for WSNs with sinks, i.e. Length-constrained Connectivity and Rerouting Centrality (l-CRC). The centrality variant measures the importance of a vertex v based on the impact of removing v, which affects the connectivity of the network and the path length of other vertices whose shortest paths to the sinks originally pass v. l-CRC is a 2-tuple index, which consists of a pair of centrality values: Length-constrained Connectivity Centrality (l-CC) and Length-constrained Rerouting Centrality (l-RC). The former is concerned with network connectivity, while the latter is with the additional length of the shortest paths. Formally, we define l-CRC of a vertex v as

$$l\text{-CRC}(v) = < l\text{-CC}(v), l\text{-RC}(v) > \tag{2}$$

6.2.1 Length-constrained Connectivity Centrality (l-CC)

The length-constrained connectivity centrality of a vertex v is the number of v's descendants that would be either disconnected or pushed over the path length limit $l_{\max}$ when v is removed from the network. We define l-CC(v), the length-constrained connectivity centrality of a vertex v as

$$l\text{-CC}(v) = |\{w \in D(v); \quad d(w, Sink) \leq l_{\max}, d_v(w, Sink) > l_{\max}\}| \tag{3}$$

where $D(v)$ is the set of v's descendants in the routing tree that are sensor nodes, not relays. $d(w, Sink)$ denotes the shortest path length between w and $Sink$, while $d_v(w, Sink)$ represents the length of the shortest path from w to $Sink$ which does not visit v.

To compare the relative l-CC of vertices from different graphs, it is desirable to have a measure that is independent of network size. A vertex v can at most disconnect $n-1$ other vertices in a graph, excluding itself. Therefore, the relative l-CC of any vertex v in a graph may be expressed as a ratio

$$l\text{-CC}'(v) = \frac{|\{w \in D(v); \quad d(w, Sink) \leq l_{\max}, d_v(w, Sink) > l_{\max}\}|}{n-1} \tag{4}$$

Algorithm 8: Vertex-Rerouting

Input : w, G, d, S
Output: $d_v(w, Sink)$

```
1: d_v(w, Sink) ← ∞
2: for all u ∈ N(w) do
3:    if u ∈ S and d(u, Sink) < d_v(w, Sink) then
4:       d_v(w, Sink) ← d(u, Sink) + 1
5:       S ← S ∪ {w}
6:    end if
7: end for
8: return d_v(w, Sink)
```

The relative centrality score has been normalised to take value in the interval [0, 1] because it is divided by the maximum possible score in networks of equal size (number of vertices). A value close to one would mean that the vertex is important to network's connectivity and path lengths of other vertices. Likewise, a value close to zero would mean that it is not important.

We measure the relative length-constrained connectivity centrality of a vertex v by basically reconstructing the routing tree for all descendants of v starting from the direct children of v. We firstly give the pseudocode for vertex rerouting in Algorithm 8, then we present the pseudocode to compute the relative l-CC in Algorithm 9.

Algorithm 8 takes as input a vertex w that needs to find a new route, the graph $G = (V, E)$, the distances d of all vertices to the sinks and a set S of vertices that w can visit when it finds a new route. In Vertex-Rerouting, w finds a new route by finding a neighbour that is in S and has the shortest route to the sinks. If such a neighbour exists, w has a new routing path and is added to S.

The algorithm to compute the relative l-CC is presented in Algorithm 9. The inputs to the algorithm are the vertex v to be measured, v's descendants list $D(v)$ in increasing order of distances to the sinks, the graph $G = (V, E)$, the distances d of all vertices to the sinks and the maximum path length limit $l_{\max}$. The descendants list, the distances and the parents of all vertices can be obtained by one run of breadth first search for unweighted graphs or Dijkstra's algorithm for weighted graphs. In this thesis, we model the WSNs as unweighted graphs. Therefore, we

Algorithm 9: l-CC

Input : $v, D(v), G, d, l_{\max}$
Output: C'

1: $S \leftarrow V \setminus (D(v) \cup \{v\})$
2: $C' \leftarrow 0$
3: **for** all $w \in D(v)$ in increasing order of $d(w, Sink)$ **do**
4: $d_v(w, Sink) \leftarrow$ Vertex-Rerouting(w, G, d, S)
5: **if** $d(w, Sink) \leq l_{\max}$ **and** $d_v(w, Sink) > l_{\max}$ **then**
6: $C' \leftarrow C' + 1$
7: **end if**
8: **end for**
9: $C' \leftarrow \frac{C'}{n-1}$
10: **return** C'

use breadth first search to build the routing tree which gives us the shortest routes from all vertices to the sinks.

In line 1 of Algorithm 9, we try to distinguish the subtree rooted at v by not including it in S. Then, in line 4, each descendant w of v finds a new route by utilising Vertex-Rerouting in Algorithm 8. In line 5 to 7, we calculate l-CC using Equation (3). The relative l-CC is obtained in line 9 using Equation (4).

6.2.2 Length-constrained Rerouting Centrality (l-RC)

The length-constrained rerouting centrality of a vertex v is the total percentage of additional length of the shortest paths, which are over the path length limit $l_{\max}$, from v's descendants to the sinks upon removal of v. Note that we only take v's descendants that are still connected to the routing tree after v is removed, because the sum of distances is only meaningful for a connected graph. We define l-RC(v), the length-constrained rerouting centrality of a vertex v as

$$l\text{-RC}(v) = \sum_{w \in D(v),\ d_v(w, Sink) \neq \infty} \left(\frac{\max\{d_v(w, Sink), l_{\max}\}}{\max\{d(w, Sink), l_{\max}\}} - 1 \right) \tag{5}$$

Given a vertex v, the shortest length of a path a vertex w has to the sink passing through v is two and the longest length of a path bypassing v is n–1. By assumption that $2 \leq l_{\max} \leq n$–1, the maximum of l-RC(v) is $|D(v)| \times \left(\frac{n-1}{l_{\max}} - 1 \right)$. Then, the relative

Algorithm 10: l-RC

Input : $v, D(v), G, d, l_{\max}$
Output: C'

$S \leftarrow V \setminus (D(v) \cup \{v\})$
$C' \leftarrow 0$
for all $w \in D(v)$ in increasing order of $d(w, Sink)$ **do**
  $d_v(w, Sink) \leftarrow$ Vertex-Rerouting(w, G, d, S)
  if $d_v(w, Sink) \neq \infty$ **then**
    $C' \leftarrow C' + (\frac{\max\{d_v(w,Sink), l_{\max}\}}{\max\{d(w,Sink), l_{\max}\}} - 1)$
  end if
end for
$C' \leftarrow \frac{l_{\max} \times C'}{|D(v)| \times (n-1-l_{\max})}$
return C'

l-RC of a vertex v is defined as

$$l\text{-RC}'(v) = \frac{l_{\max}}{|D(v)| \times (n-1-l_{\max})} \sum_{w \in D(v),\ d_v(w,Sink) \neq \infty} \left(\frac{\max\{d_v(w, Sink), l_{\max}\}}{\max\{d(w, Sink), l_{\max}\}} - 1 \right) \tag{6}$$

Values of l-RC$'(v)$ may be compared between graphs. Algorithm 10 calculates the relative l-RC using Equation (6).

6.2.3 l-CRC Ranking

We rank sensors based on l-CRC to identify the top-rank critical sensors that need backups. So, if we are given a tight budget that can only protect m critical sensors out of n total sensors in a network, we can easily select the first m sensors in the ranking list. Because l-CRC is a 2-tuple index, we have two choices to rank the centrality scores, i.e. based on l-CC scores or l-RC scores. We called the first choice the *primary order* and the second choice the *secondary order*. Although the primary order is more important than the secondary order, at a certain point below a specific value of the primary order, the secondary order may become more important. Therefore, we define two threshold values for the primary and secondary orders, namely the *primary threshold* and the *secondary threshold*. The ranking process works as follows:

1. The rank is based on the primary order from the highest primary order's value down to the primary threshold. If two or more primary order's values are the same, the rank is chosen based on the secondary order's value, where the highest comes first.

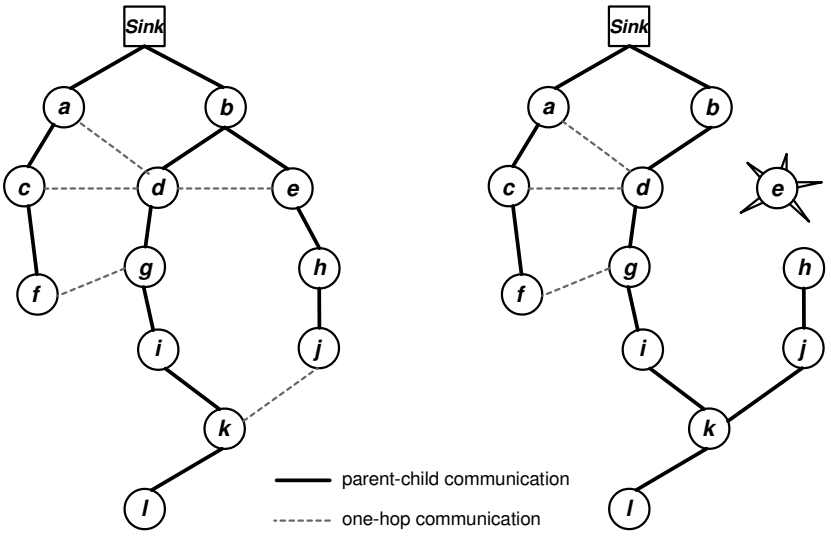
2. After the primary threshold, the rank is based on the secondary order from the highest one remaining down to the secondary threshold. If two or more secondary order's values are the same, the rank is chosen based on the primary order's value.

3. After the secondary threshold, the vertex rank is chosen arbitrarily, where the highest one remaining from either the primary or the secondary order comes first.

Based on the topology planning objectives, one may choose l-CC as the primary order and l-RC as the secondary order, or vice versa. In this thesis, we assume that l-CC is more important than l-RC because it takes into account the number of disconnected vertices. So, we take l-CC as the primary order and l-RC as the secondary order. We assign the *Connectivity Centrality Threshold* (CT) as the primary threshold and the *Rerouting Centrality Threshold* (RT) as the secondary threshold. The Connectivity Centrality Threshold (CT) is the tolerable percentage of a network that will be effectively disconnected after the failure of one vertex. By analogy, the Rerouting Centrality Threshold (RT) specifies a tolerance on the extra length of the shortest paths. If both thresholds are 0%, we are trying to achieve 2-connectivity for vertices which are two or more hops away from the sinks.

6.2.4 l-CRC Example

Figure 74(a) is an example of a small network which consists of 12 sensors and one sink with $l_{\max}=5$. Note that in this topology, $d(l, Sink) > l_{\max}$. The solid lines are the original routing paths to the sink while the dashed lines show the communication



Node	C_D'
d	0.4167
a	0.25
b	0.25
c	0.25
e	0.25
g	0.25
k	0.25
f	0.1667
h	0.1667
i	0.1667
j	0.1667
l	0.0833

(c)

Node	C_B'
d	0.3914
g	0.3081
i	0.2172
e	0.2071
k	0.1667
h	0.1465
j	0.1010
b	0.0884
a	0.0631
c	0.0606
f	0.0303
l	0

(d)

Node	*l*-CRC′
e	<0.1818, 0.25>
h	<0.0909, 0.1667>
g	<0.0909, 0.0556>
a	<0, 0>
b	<0, 0>
c	<0, 0>
d	<0, 0>
f	<0, 0>
i	<0, 0>
j	<0, 0>
k	<0, 0>
l	<0, 0>

(e)

Figure 74: l-CRC example. (a) is the original paths to the sink, (b) is the alternative paths if vertex e dies, (c) shows the relative scores for degree centrality (C'_{D}) for (a), (d) shows the relative scores for betweenness centrality (C'_{B}) for (a), and (e) shows the l-CRC′ scores for (a) with $l_{\max}=5$.

links. Before describing the relative scores of length-constrained connectivity and rerouting centrality (l-CRC$'$), we will firstly show the relative scores for degree centrality ($C_{\mathrm{D}}{}'$) and betweenness centrality ($C_{\mathrm{B}}{}'$).

The formulas to calculate degree centrality and betweenness centrality have been presented in Section 2.6.1. Recall that the degree centrality of a vertex v is the number of vertices adjacent to v. To normalise the score, we divide it by the maximum possible score of degree centrality, i.e. $n-1$ [44]. The betweenness centrality of a vertex v is the sum of the probability that v falls on a randomly selected shortest path between all pairs of vertices in the graph. We divide the score by $\frac{(n-1)(n-2)}{2}$ to normalise it [44]. Note that since the calculations of degree centrality and betweenness centrality do not differentiate between a sink and a sensor, we assume that the sink in Figure 74(a) is an ordinary vertex when we calculate $C_{\mathrm{D}}{}'$ and $C_{\mathrm{B}}{}'$, so the number of vertices in the graph is 13.

According to the $C_{\mathrm{D}}{}'$ and $C_{\mathrm{B}}{}'$ scores as shown in Figure 74(c) and (d), respectively, the most central vertex is d. However, l-CRC$'$ for d is $<0,0>$ as shown in Figure 74(e). It means, d is not critical because its descendants have length-bounded alternative paths, which are not longer than $l_{\max}$. Hence, both degree centrality and betweenness centrality would not be useful in this case. Vertices e, h and g have l-CRC$'$ equal to $<0.1818, 0.25>$, $<0.0909, 0.1667>$ and $<0.0909, 0.0556>$, respectively, since at least one of their descendants' alternative paths are longer than $l_{\max}$. Vertex k, on the other hand, has l-CRC$'$ equal to $<0,0>$, because even if l is disconnected upon its removal, l's original path is already longer than $l_{\max}$. The routing paths when e fails are illustrated in Figure 74(b). In Figure 74(e), l-CRC$'$ scores are ranked using l-CC$'$ as the primary order.

We have presented the concept of l-CRC to measure the importance of each sensor node in a network. We will use l-CRC to identify critical sensors and to assess the quality of relay deployment positions to provide alternative paths around the most critical ones. In the following section, we will describe our local search algorithm to deploy relays.

6.3 Greedy Randomised Adaptive Search Procedure for Additional Backup Placement (GRASP-ABP)

We consider WSNs where the vertices are partitioned into sensors, backup nodes (relays) and sinks, and so in the graph representation $V = T \cup B \cup S$. We define the general *additional backup placement problem*: given a graph $G = (T \cup A \cup S, E)$, where A is a set of candidate positions for relays, find a minimal subset $B \subseteq A$ so that when a sensor $v \in T$ in a graph $H = (T \cup B \cup S, E\downarrow_{T \cup B \cup S})$ dies, each of its descendants has an alternative path to a sink $s \in S$ of length $\leq l_{\max}$.

Specifically, for the *additional backup placement with centrality thresholds problem*, given a sensor $v \in T$ with $l\text{-CRC}(v) = < l\text{-CC}(v), l\text{-RC}(v) >$, CT is the primary threshold for l-CC, and RT is the secondary threshold for l-RC, it is required that each sensor $v \in T$ in the graph $H = (T \cup B \cup S, E\downarrow_{T \cup B \cup S})$ has $l\text{-CC}(v) \leq$ CT and $l\text{-RC}(v) \leq$ RT.

We propose the GRASP algorithm for the additional backup placement (GRASP-ABP) to deploy a minimum number of additional relays for ensuring the existence of alternative paths in a routing tree. Before the execution of GRASP-ABP, we need to determine the set of critical vertices X and the set of non-critical vertices Y, which become inputs to this algorithm. For this matter, we calculate $l\text{-CRC}(v)$ for each $v \in T$ in an graph $(T \cup S, E\downarrow_{T \cup S})$. If either $l\text{-CC}(v) >$ CT or $l\text{-RC}(v) >$ RT, v is critical and we put it in X. Finally, $Y = T \setminus X$.

6.3.1 Construction Phase

The first step in any GRASP algorithm is to construct an initial solution. The initial solution in GRASP-ABP is B, an initial set of backup nodes (relays). Each relay in B is identified by finding the shortest path from each descendant, that is a

sensor, of each critical vertex to a sink in G bypassing each critical vertex. For each relay found in the shortest path, we put it in B. The randomisation of the initial solution is obtained by randomly selecting parents in the shortest paths if there are hop count ties. At the end of the construction phase, a graph $H = (T \cup B \cup S, E{\downarrow}_{T \cup B \cup S})$ is generated.

6.3.2 Node-based Local Search

The next stage in a GRASP algorithm is to explore the neighbourhood of the initial solution, looking for lower cost solutions. Let B be the set of backup nodes (relays) in the current graph H. We explore the neighbourhood of the current solution by adding a new relay $r \in A \backslash B$ into B that can eliminate as many existing relays from B as possible.

6.3.3 Algorithm Description

We describe GRASP-ABP that consists of the construction phase and the node-based local search phase. The pseudocode for GRASP-ABP is given in Algorithm 11. It takes as input the original graph $G = (T \cup A \cup S, E)$, the set S of sinks, the set A of candidate backups, the set X of critical vertices, the set Y of non-critical vertices, the descendants of the critical vertices $D(X)$, the connectivity centrality threshold CT, the rerouting centrality threshold RT, the maximum acceptable path length $l_{\max}$, and the number of iterations (*max_iterations*). The procedure is repeated *max_iterations* times. In each iteration, a greedy randomised solution to find an initial set of relays B is executed from line 3 to 9. The current graph $H = (T \cup B \cup S, E{\downarrow}_{T \cup B \cup S})$ is generated in line 10 and the routing tree of H is built in line 11. The l-CRC scores for the critical vertices are then recalculated in line 12 using the routing tree of H to obtain the highest values of l-CC and l-RC, denoted $l\text{-CC}_{\max}(H)$ and $l\text{-RC}_{\max}(H)$, respectively. Note that we only calculate the l-CRC score for each critical vertex $v \in X$, which is a sensor. v may have both sensors and relays as its

Algorithm 11: GRASP-ABP

Input : $G, S, A, X, Y, D(X), CT, RT, l_{\max}, max\_iterations$
Output: B^*

1: $best\_value \leftarrow \infty$
2: **for** $i \leftarrow 1$ to $max\_iterations$ **do**
/* Construction phase */
3: $B \leftarrow \emptyset$
4: **for** all $v \in X$ **do**
5: **for** all $w \in D(v)$ **do**
6: Find the shortest path from w to a sink $s \in S$ bypassing v
7: $B \leftarrow B \cup \{r\}$ for each $r \in A$ found in the shortest path
8: **end for**
9: **end for**
10: $H \leftarrow (X \cup Y \cup B \cup S, E{\downarrow_{X \cup Y \cup B \cup S}})$
11: Build the routing tree of H
12: Calculate l-$\mathrm{CC}_{\max}(H)$ and l-$\mathrm{RC}_{\max}(H)$
/* Local search phase */
13: **do**
14: $solution\_updated \leftarrow$ **false**
15: $best\_set \leftarrow B, \quad best\_number \leftarrow |B|$
16: **for** all $r \in A \setminus B$ **do**
17: $L \leftarrow \emptyset$
18: **for** all $t \in B$ **do**
19: $L \leftarrow L \cup \{t\}$
20: Build the routing tree of H^{+r-L}
21: Calculate l-$\mathrm{CC}_{\max}(H^{+r-L})$ and l-$\mathrm{RC}_{\max}(H^{+r-L})$
22: **if** (l-$\mathrm{CC}_{\max}(H^{+r-L}) > l$-$\mathrm{CC}_{\max}(H)$ **or** l-$\mathrm{RC}_{\max}(H^{+r-L}) > l$-$\mathrm{RC}_{\max}(H)$) **and** ($l$-$\mathrm{CC}_{\max}(H^{+r-L}) > CT$ **or** l-$\mathrm{RC}_{\max}(H^{+r-L}) > RT$) **then**
23: $L \leftarrow L \setminus \{t\}$
24: **end if**
25: **end for**
26: **if** $|B| - |L| + 1 < best\_number$ **then**
27: $best\_set \leftarrow (B \cup \{r\}) \setminus L, \quad best\_number \leftarrow |B| - |L| + 1$
28: **end if**
29: **end for**
30: **if** $best\_number < |B|$ **then**
31: $B \leftarrow best\_set, \quad H \leftarrow (X \cup Y \cup B \cup S, E{\downarrow_{X \cup Y \cup B \cup S}})$
32: $solution\_updated \leftarrow$ **true**
33: **end if**
34: **while** $solution\_updated$
/* Best solution update */
35: **if** $|B| < best\_value$ **then**
36: $B^* \leftarrow B, \quad best\_value \leftarrow |B|$
37: **end if**
38: **end for**
39: **return** B^*

descendants in the routing tree. However, for the l-CRC calculation, we only take into account the descendants of v that are sensors, not relays. We do not provide alternative paths for relays because they are deployed only as additions to support connectivity.

The local search starts with the initialisation of the best set and the best number of backups in line 15. The loop from line 16 to 29 searches for the best move, i.e. finding a new relay $r \in A \setminus B$ that can eliminate as many existing relays from B as possible. The loop from line 18 to 25 tries to find the maximum set $L \subseteq B$ of existing relays that are safe to be removed after the insertion of r. To check if the relays in L are safe to be removed, firstly we build the routing tree of H^{+r-L} in line 20 and calculate the new highest values of l-CC and l-RC using the routing tree of H^{+r-L} in line 21, namely $l\text{-CC}_{\max}(H^{+r-L})$ and $l\text{-RC}_{\max}(H^{+r-L})$. Then, in line 22, we check if the new scores improve the previous ones or stay below the thresholds. If the new solution improves the number of backups used in the current best solution, the best set and the best number of relays are updated in line 27. When all moves have been evaluated, we check in line 30 if an improving solution has been found. If the moves produce a better solution, the set of relays B is updated in line 31, as well as the graph H. Then, the local search continues.

If, at the end of the local search, we found a better solution compared to the best solution found so far, we update in line 36 the set of relays and the least number of backups used. The best set B^* of relays is returned in line 39.

6.4 Evaluation of GRASP-ABP

We evaluate the effectiveness and the efficiency of GRASP-ABP, which uses l-CRC, compared to some existing closely related algorithms that are used in planning a network. We use these metrics, which have been discussed in Chapter 3, to measure the performance of the algorithms:

1. ***Number of additional relay nodes* (cost)**. We compare the effectiveness of the algorithms by using the total number of deployed relay nodes in the network.

2. ***Runtime***. We measure the efficiency of the algorithms by showing their computation time.

3. ***Number of disjoint paths***. We want to compare the average number of disjoint paths per node after we deploy relays in the network.

4. ***Percentage of nodes with disjoint paths***. We want to show the relationship between deploying fewer relays and the number of nodes that have disjoint paths.

We compare GRASP-ABP against GRASP-ARP from Chapter 5 and K-CONN-REPAIR [90] that has been modified for constrained deployment locations, where relay nodes can only be placed in specific candidate locations. All algorithms are written in C++. Our simulation results are based on the mean value of 20 randomly generated network deployments. The network consists of up to 100 nodes deployed within randomly perturbed grids of a two-dimensional area, where a node is placed in a unit grid square of 8m $\times$ 8m and the coordinates are perturbed. In order to get sparse networks (average degree 2–3), we generate more grid points than the number of nodes. For example, we use 6 $\times$ 6, 8 $\times$ 8 and 11 $\times$ 11 grid squares to randomly deploy 25, 49 and 100 nodes, respectively. Candidate relays are also distributed in a grid area, where a candidate occupies a unit grid square of 6m $\times$ 6m. Both sensor and relay nodes use the same transmission range, i.e. 10 metres.

In the simulation, we use 25-node, 49-node and 100-node topologies with 49, 100 and 196 candidate relays, respectively. The maximum path length ($l_{\max}$) is set to 10 for 25-node, 15 for 49-node and 20 for 100-node networks. The maximum number of iteration of GRASP is 10. We use connectivity as our primary order in centrality score ranking and rerouting as secondary. The thresholds for connectivity (CT) and rerouting (RT) are both 0% and 2%. If CT and RT are 0%, we are trying to achieve

2-connectivity for nodes which are two or more hops away from the sink. Therefore, we only simulate 2-connectivity for GRASP-ARP and K-CONN-REPAIR. If CT and RT are greater than 0%, we trade-off the number of relay nodes (cost) and runtime against the quality of designed networks.

6.4.1 Multiple Sources – Single Sink Problem

In this simulation, we put the sink at the top-left corner of the network, while all sensor nodes are the source nodes. We simulate two versions of GRASP-ARP, i.e. using the basic version of Counting-Paths and its dynamic programming variant (DP).

Figure 75 shows the number of additional backup nodes (relays) needed. On average, GRASP-ABP finds the least number of additional relay nodes compared to K-CONN-REPAIR and GRASP-ARP. GRASP-ABP deploys fewer relays than GRASP-ARP as it does not need to provide alternative paths for sensor nodes which are connected directly to the sink and have no descendants. Furthermore, an alternative path is not sought for a sensor node that only has one neighbour and has an original path that is already longer than $l_{\max}$. This node has no influence on its parent's l-CRC score. The algorithms' runtime for the multiple sources – single sink problem is shown in Table 15, where GRASP-ABP's runtime is 3.5 times faster than K-CONN-REPAIR and 2.8 times faster than GRASP-ARP-DP in 100-node networks. When the thresholds are increased from 0% to 2%, GRASP-ABP deploys 40% fewer relays and reduces the computational times.

We then use the Counting-Paths algorithm from Chapter 5 to find the number of disjoint paths and the number of nodes that have disjoint paths in the topologies generated by the backup placement algorithms. All sensor nodes in the topologies generated by K-CONN-REPAIR, GRASP-ARP-DP and GRASP-ARP have 2 disjoint paths. Therefore, the average number of disjoint paths found is two and the

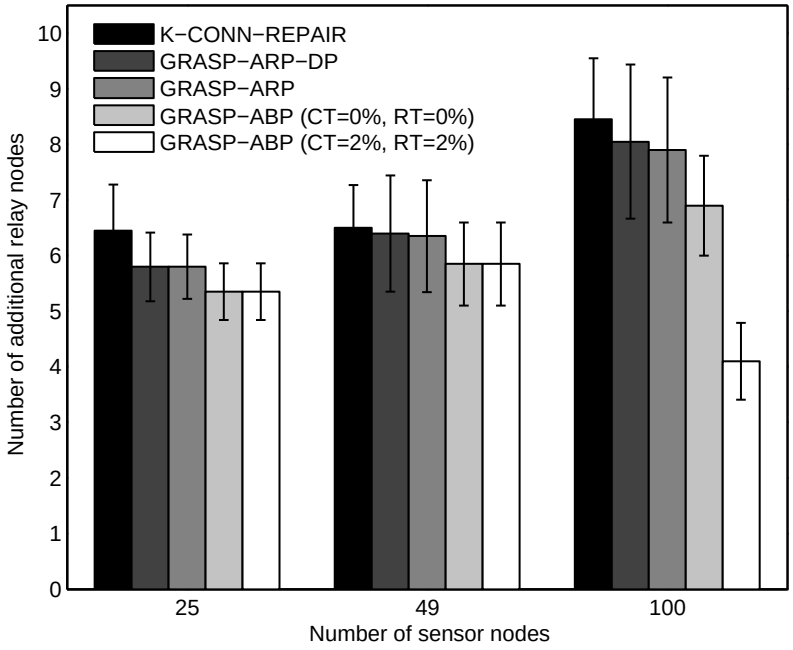


Figure 75: Number of backup nodes needed versus number of sensor nodes for multiple sources – single sink

Table 15: Additional backup placement algorithms' runtime for multiple sources – single sink

Algorithms	**Runtime (sec)**		
	25-node	**49-node**	**100-node**
K-CONN-REPAIR	6.2891	254.7343	10,003.8000
GRASP-ARP-DP	24.3469	421.2820	7,897.9650
GRASP-ARP	39.4860	619.3118	13,964.9400
GRASP-ABP (CT=0%, RT=0%)	5.9163	132.0446	2,830.4940
GRASP-ABP (CT=2%, RT=2%)	5.9095	134.5218	2,207.5260

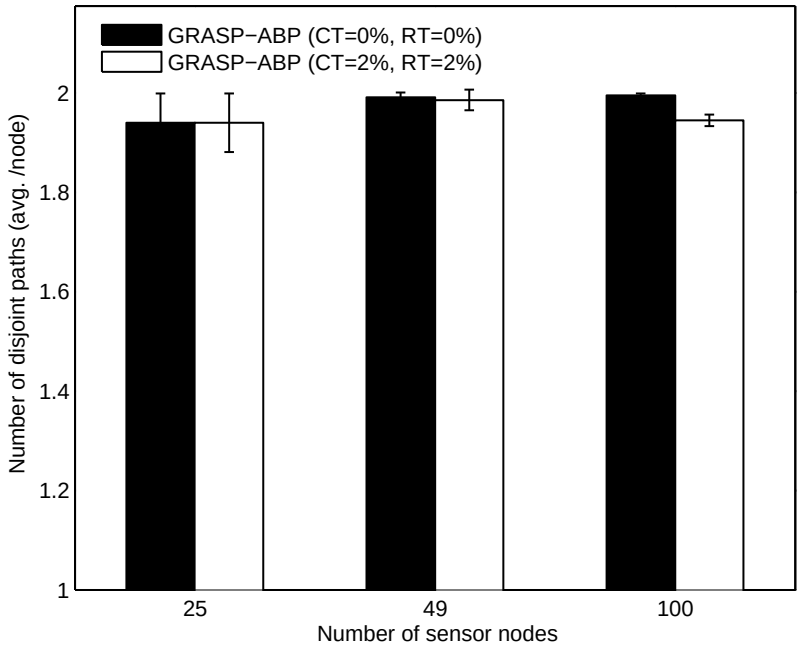


Figure 76: Number of disjoint paths found for multiple sources – single sink in GRASP-ABP topologies

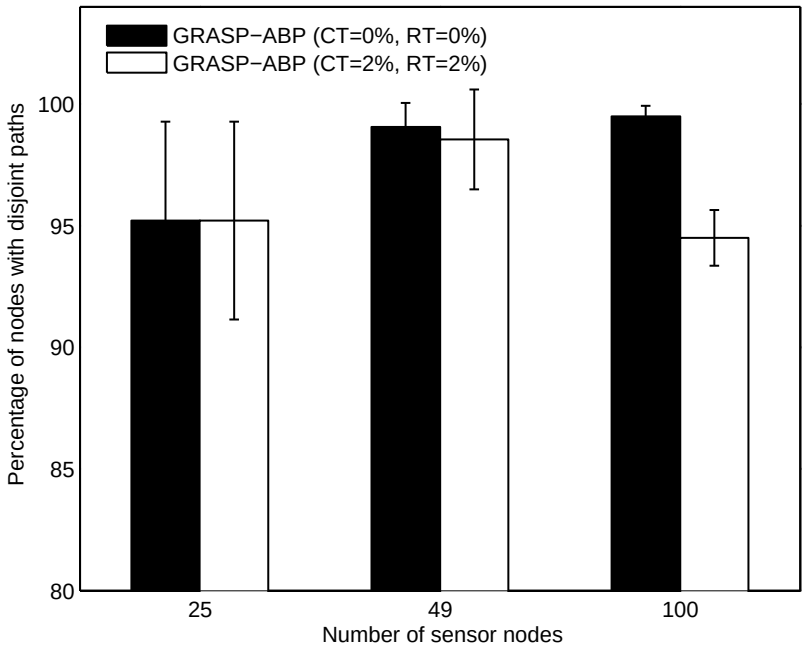


Figure 77: Percentage of nodes with disjoint paths for multiple sources – single sink in GRASP-ABP topologies

Table 16: Additional backup placement algorithms' runtime for multiple sources – multiple sinks

Algorithms	Runtime (sec)		
	25-node	**49-node**	**100-node**
K-CONN-REPAIR	6.2891	254.7343	10,003.8000
GRASP-ARP-AnySinks-DP	1.7844	37.7884	517.3695
GRASP-ARP-AnySinks	2.5180	55.3843	735.7915
GRASP-ARP-DiffSinks-DP	1.6351	31.3648	426.6710
GRASP-ARP-DiffSinks	2.3828	51.4437	822.0718
GRASP-ABP (CT=0%, RT=0%)	1.0540	7.7484	85.4845
GRASP-ABP (CT=2%, RT=2%)	1.0509	6.6265	37.2608

percentage of nodes with disjoint paths is 100%. Figure 76 and 77 show the simulation results for the GRASP-ABP topologies. Firstly, these two figures show that in the GRASP-ABP topologies, not all sensor nodes have 2 disjoint paths because the topologies have less number of relay nodes. Secondly, as the thresholds increase from 0% to 2%, the number of sensor nodes that have 2 disjoint paths decrease because the networks have fewer relays, and so the average number of disjoint paths and the percentage of nodes with disjoint paths slightly drop as shown in Figure 76 and 77.

6.4.2 Multiple Sources – Multiple Sinks Problem

We have four sinks deployed at the top-left, top-right, bottom-left and bottom-right corners of the network. Recall that in the multiple sink problem, there are two cases for interpreting the disjoint path requirement: *different-sinks* and *any-sinks*. The different-sinks problem is where original and backup paths must terminate at different sinks. The any-sinks problem is the case where the paths may terminate at any sinks. GRASP-ABP only provides solutions for the any-sinks problem as our centrality calculations do not put any restrictions on where a node's alternative path must terminate at. When a node finds an alternative path, it selects a new parent from its neighbourhood that has the shortest path length to a sink, which may be the same or different to its previously connected sink.

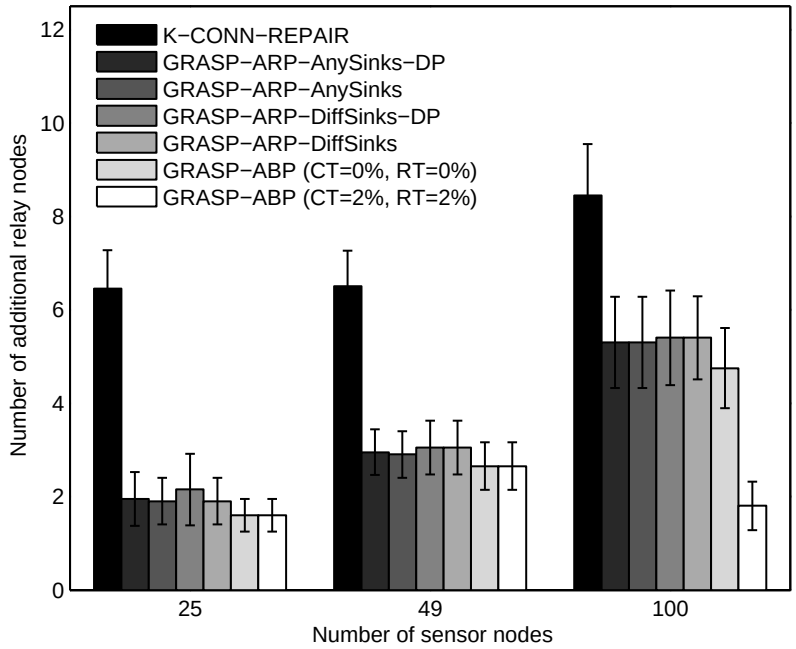


Figure 78: Number of backup nodes needed versus number of sensor nodes for multiple sources – multiple sinks

Figure 78 and Table 16 show our simulation results, where GRASP-ABP outperforms GRASP-ARP and K-CONN-REPAIR in both the number of additional relay nodes needed and the runtime. Compared to the single sink case, the GRASP algorithms have shorter runtime when there are many sinks in the network. This happens because sinks do not have many descendants in their routing tree and the paths from sensor nodes to the sinks are shorter, so the shortest path computation time is faster. GRASP-ABP requires almost 20% fewer relays and runs more than six times faster than GRASP-ARP with dynamic programming for the case of 100-node networks. In addition, by using 2% threshold, GRASP-ABP provides cheaper and faster answers to the same problem. K-CONN-REPAIR has the worst performance in relay deployment process as it must run an expensive connectivity checking algorithm in each iteration to provide k-connectivity for an entire network.

After we deploy relays, we run Counting-Paths to find the average number of disjoint paths per node and the percentage of nodes that have disjoint paths in each

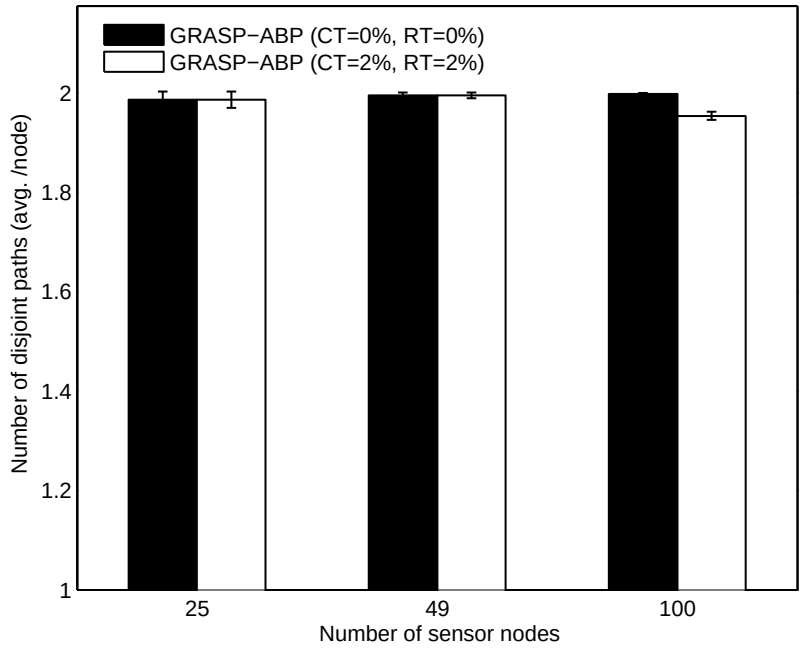


Figure 79: Number of disjoint paths found for multiple sources – multiple sinks in GRASP-ABP topologies

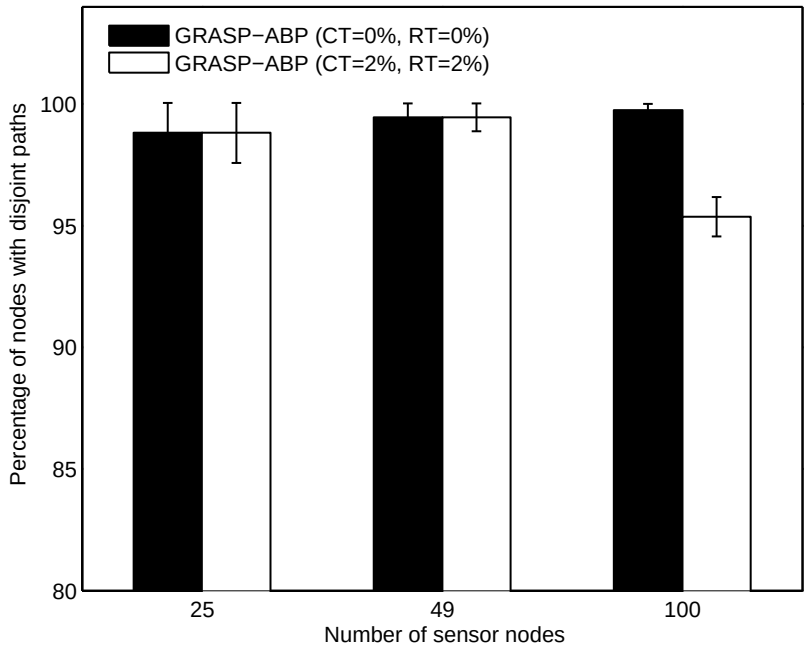


Figure 80: Percentage of nodes with disjoint paths for multiple sources – multiple sinks in GRASP-ABP topologies

generated topology. Similar to the single sink case, all sensor nodes in the topologies generated by K-CONN-REPAIR, GRASP-ARP-DP and GRASP-ARP for both any-sinks and different-sinks problems have 2 disjoint paths. Hence, the average number of disjoint paths found is two and the percentage of nodes with disjoint paths is 100%. Figure 79 and 80 show the results for the GRASP-ABP topologies. These results also indicate similar trends to the single sink scenario, where not all sensor nodes in the GRASP-ABP topologies have 2 disjoint paths, especially when the thresholds for connectivity and rerouting are 2%.

6.5 Conclusion

To be robust to failures, network topologies should provide alternative routes to the sinks so when failures do occur the routing protocol can still offer reliable delivery. Our contribution is a solution that can achieve such reliability in a more efficient manner than other published approaches. We ensure that each sensor node in the initial design has an alternative path to the sinks by deploying a small number of additional backup nodes (relays). To solve this problem, we define l-CRC, a new centrality measure that determine a node's importance to connectivity and efficient delivery in the network. We use l-CRC scores to identify the most important nodes and to provide alternative paths around those nodes. We also introduce GRASP-ABP, a local search algorithm to be run during the initial topology planning to minimise the number of relays that need to be deployed.

We evaluate the algorithm in terms of the number of additional relays it deploys and its runtime, where we demonstrate that the centrality-based GRASP-ABP's deploys fewer additional relays with faster runtime compared to the other algorithms. This is achieved by providing 2-connectivity only for sensor nodes which are two or more hops away from a sink and whose original shortest paths are length-bounded. GRASP-ABP obviously retreats from the network-wide 2-connectivity and having higher thresholds retreats even further. So, we would expect it to be faster but

poorer on the disjoint path metric. To show that the topologies generated by GRASP-ABP with 0% and 2% thresholds have comparable performance to the topologies which have more relays, we will evaluate the network performance for each topology in Chapter 8. We will take the topologies generated by the topology planning algorithms, deploy sensor nodes, relay nodes and sinks according to the deployment plans, and evaluate their performances while some nodes are failing. By simulating the network operation, we want to show the trade-off between the efficiency of the network design and the robustness of the network performance.

Chapter 7

Multiple Sink and Relay Placement

7.1 Introduction

In this chapter, we define a novel problem for increasing the Wireless Sensor Network (WSN) topology robustness against a single failure by deploying multiple sinks and relays with minimal cost. Our solution differs from the ones in the literature, which were reviewed in Section 2.7, because we consider both sink and relay deployment at the same time for fault-tolerant multi-hop networks with a path length constraint. A network is robust against a single failure if after a failure of a sink, a sensor node or a relay node, each remaining sensor node can deliver its data to a sink through a multi-hop path with an acceptable length. To be robust to sink failure, we deploy multiple sinks in the network such that each sensor node is *double-covered*, i.e. it has length-bounded paths to two sinks. While we restrict our assumption to k-covered, where $k=2$ in this chapter, our solution is also applicable to any integer $k \geq 1$. If $k > 2$, the solution is robust to multiple (up to $k-1$) sink failures. To protect against sensor node failure, we place some relay nodes so that every sensor node is *non-critical*, i.e. when it fails, each remaining sensor node still has at least

one length-bounded path to a sink. Since installing multiple sinks and relays incurs additional costs, our solution tries to minimise the total deployment cost. We solve the multiple sink and relay placement problem using a greedy algorithm and a local search algorithm based on GRASP [41, 42, 96].

Firstly, we look at the multiple sink placement (MSP) problem to minimise the number of *uncovered* nodes, i.e. nodes that are not double-covered. We present Greedy-MSP and GRASP-MSP in Section 7.2 and show the simulation results in Section 7.3. The simulation results show that Greedy-MSP has the shortest runtime but deploys more sinks than GRASP-MSP. On the other hands, our simulation demonstrates that GRASP-MSP finds comparable cost to the optimal solution with shorter runtime than our implementation of the optimal solution. Even though finding the optimal solution is sufficient for the multiple sink placement problem, the GRASP-MSP performance gives us confidence to use the same local search technique for the more complex multiple sink and relay placement problem because it can achieve almost the same solution as the optimal one and even faster.

After that, we look at the multiple sink and relay placement (MSRP) problem and present Greedy-MSRP and GRASP-MSRP in Section 7.4. Both algorithms employ the concept of Length-constrained Connectivity and Rerouting Centrality (l-CRC) introduced in Chapter 6 to identify every *critical* node, i.e. a sensor node which if fails can cause other nodes to lose their length-bounded paths to sinks. Greedy-MSRP deploys sinks and relays separately. It basically uses Greedy-MSP to minimise the number of uncovered nodes by deploying multiple sinks. Since in this thesis we assume that the cost of a sink is more expensive than a relay node because it is usually powered, has large storage capacity and has WiFi/ethernet backhaul, Greedy-MSRP tries to trade some sinks for relays to minimise the total deployment cost but ensures that the network is still double-covered and non-critical. To be used by Greedy-MSRP for multiple relay placement (MRP), we develop GRASP-MRP. GRASP-MRP extends GRASP-ABP in Chapter 6 to make a network topology not only non-critical, but also double-covered. Unlike Greedy-MSRP, GRASP-MSRP

minimises the number of uncovered and critical nodes simultaneously in its every local search move. We demonstrate empirically in Section 7.5 that GRASP-MSRP runs faster than Greedy-MSRP and the solutions produced by GRASP-MSRP are over 30% less costly than those of Greedy-MSRP.

7.2 Multiple Sink Placement (MSP)

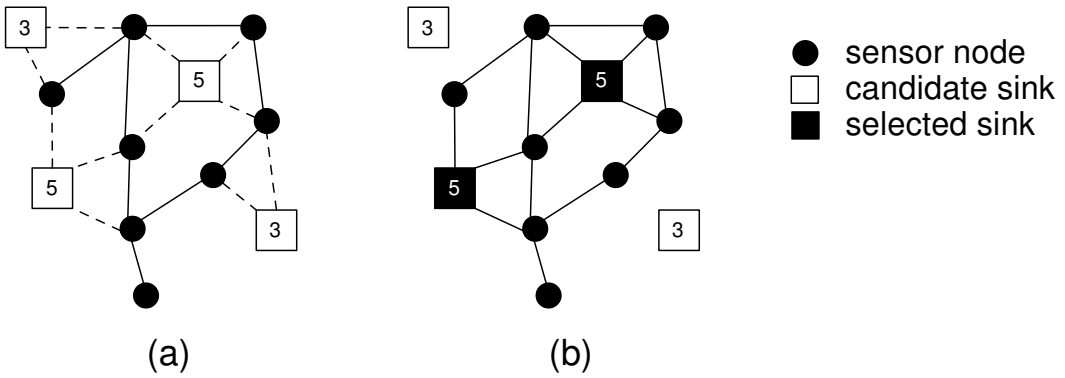


Figure 81: Illustration of the MSP problem. (a) A WSN with four candidate sinks and (b) the double-covered WSN where $l_{\max}=3$.

We partition vertices into a set of sensors T and sinks S. In the graph representation of a WSN, $V=T\cup S$. Note that at this stage we do not use relays yet. A sensor is *double-covered* if and only if it has at least two paths of length $\leq l_{\max}$ to two sinks in S. If a sensor is not double-covered, it is *uncovered*. We define a WSN as double-covered if each sensor $v\in T$ is double-covered. In the *multiple sink placement problem*, given a graph $G=(T\cup A_{\mathrm{S}}, E)$, where A_{S} is a set of candidate locations for sinks with a non-negative cost function $c: A_{\mathrm{S}}\rightarrow\mathbb{R}$, we find a minimum cost subset $S\subseteq A_{\mathrm{S}}$ such that $H=(T\cup S, E{\downarrow_{T\cup S}})$ is double-covered.

We illustrate this problem in Figure 81. Figure 81(a) illustrates a WSN with four candidate locations to deploy sinks, where the numbers represent the costs. In order to make the WSN double-covered with $l_{\max}=3$, we need to deploy the two sinks as shown in Figure 81(b). The total cost of sink deployment in this example is 10 units.

To solve the multiple sink placement problem, we present Greedy-MSP, a greedy-based algorithm, and GRASP-MSP, a local search algorithm that uses the GRASP technique [41, 42, 96]. To speed-up our algorithms' processing time, we compute the shortest path from all sensors to all candidate sinks once in the beginning and store the length of the shortest path in *Distance* table, while the parent of each sensor in the shortest path to a candidate sink is stored in *Parent* table. For example, for $G=(T\cup A_{\mathrm{S}}, E)$, $Distance_G(v, w)$ shows the length of the shortest path from a sensor $v\in T$ to a candidate sink $w\in A_{\mathrm{S}}$, while $Parent_G(v, w)$ shows the parent of a sensor v on the shortest path to a candidate sink $w\in A_{\mathrm{S}}$.

7.2.1 Greedy Algorithm for Multiple Sink Placement (Greedy-MSP)

Greedy-MSP is similar to the *Heuristic Opt Multisink Place* (HOMP) algorithm by Xu and Liang [122], but instead of deploying a minimum number of sinks to make a network single-covered, it considers double-covered networks. Greedy-MSP deploys sinks one by one until a network is double-covered. In each iteration, the greedy move picks the best sink from the set of candidate sinks that can minimise the number of uncovered sensors as possible. In Greedy-MSP, if two or more sinks offer the same number of uncovered sensors, the lowest cost sink is selected. If there are ties, we select one arbitrarily.

The Greedy-MSP pseudocode is given in Algorithm 12. It takes as input the original graph $G=(T\cup A_{\mathrm{S}}, E)$, the set T of sensors, the set A_{S} of candidate sinks, the cost function c, the pre-computed $Distance_G$ table, and the maximum acceptable path length $l_{\max}$. Initially, when there are no sinks selected, all sensors are uncovered. We initialise the set S of sink and the number of uncovered sensors in line 1. The loop from line 2 to 19 greedily selects the best sinks one by one until all sensors are double-covered. The best sink is the one that together with the previously chosen sinks can minimise the number of uncovered sensors. The loop from line 4

Algorithm 12: Greedy-MSP

Input : $G, T, A_{\mathrm{S}}, c, Distance_G, l_{\max}$
Output: S

1: $num\_uncovered \leftarrow |T|, \quad S \leftarrow \emptyset$
2: **while** $num\_uncovered > 0$ **and** $|S| < |A_{\mathrm{S}}|$ **do**
3: $max\_num\_covered \leftarrow 0$
4: **for** all $v \in A_{\mathrm{S}} \setminus S$ **do**
5: $H \leftarrow (T \cup S \cup \{v\}, E{\downarrow_{T \cup S \cup \{v\}}})$
6: **if** $|S| = 0$ **then**
7: $num\_covered_v \leftarrow$ Find $num\_single\_covered$ in H using $Distance_G$ and $l_{\max}$
8: **else**
9: $num\_covered_v \leftarrow$ Find $num\_double\_covered$ in H using $Distance_G$ and $l_{\max}$
10: **end if**
11: **if** $num\_covered_v > max\_num\_covered$ **then**
12: $max\_num\_covered \leftarrow num\_covered_v$
13: **end if**
14: **end for**
15: $best\_sink \leftarrow \{v \in A_{\mathrm{S}} \setminus S,\ num\_covered_v = max\_num\_covered, \quad \forall w \in A_{\mathrm{S}} \setminus S : c_v \leq c_w\}$
16: $S \leftarrow S \cup \{$Select a sink randomly from $best\_sink\}$
17: $H \leftarrow (T \cup S, E{\downarrow_{T \cup S}})$
18: Find $num\_uncovered$ in H using $Distance_G$ and $l_{\max}$
19: **end while**
20: **return** S

to 14 finds the maximum number of sensors that can be covered by addition of each candidate sink into the set of selected sink. If there are no sinks selected before, for each candidate sink $v \in A_{\mathrm{S}}$ we find the number of single-covered sensors, i.e. how many sensors in $Distance_G$ table that have path length $\leq l_{\max}$ to v. If there are some sinks selected before, we calculate the number of double-covered sensors. We then group together the candidate sinks that have the maximum number of covered sensors and the minimum cost in line 15. In line 16, we select one candidate sink arbitrarily to be inserted into the set of sink S. We then calculate the number of uncovered sensors in line 18. The iteration stops if all sensors are double-covered or all candidate sinks have been selected. The set of sinks S is finally returned in line 20.

7.2.2 Greedy Randomised Adaptive Search Procedure for Multiple Sink Placement (GRASP-MSP)

A greedy heuristic can easily get caught in a local minimum, where it thinks that it finds the lowest cost solution but it actually does not. GRASP is likely to be better than the greedy heuristic because the randomisation aspect in its local search enables it to escape the local minimum. We present GRASP-MSP to solve the multiple sink placement problem. As with other GRASP-based algorithms, GRASP-MSP consists of two steps: *construction phase* to construct an initial feasible solution and *local search phase* to explore the neighbourhood of the initial solution, looking for lower cost solutions.

Construction Phase

In the construction phase, we find S, an initial set of sinks. Instead of selecting the best candidate sink from A_{S} to be put in S, which can minimise the number of uncovered sensors, we add randomisation to the initial solution by choosing a sink from A_{S} randomly. This random selection is repeated until the network is double-covered or all candidate sinks have been chosen.

Node-based Local Search

Let S be the set of sinks. We explore the neighbourhood of the current solution by adding a new sink $s \in A_{\mathrm{S}} \setminus S$ into S that can eliminate some existing sinks from S to reduce the total sink cost as low as possible. This move must always ensure that the network is double-covered.

Algorithm Description

The GRASP-MSP pseudocode is given in Algorithm 13. It takes as input the original graph $G = (T \cup A_{\mathrm{S}}, E)$, the set T of sensors, the set A_{S} of candidate sinks,

Algorithm 13: GRASP-MSP

Input : $G, T, A_{\mathrm{S}}, c, Distance_G, l_{\max}, max\_iterations$
Output: S^*

1: $best\_value \leftarrow \infty$
2: **for** $i \leftarrow 1$ to $max\_iterations$ **do**
/* Construction phase */
3: Find S by choosing sinks from A_{S} randomly
4: **do**
5: $solution\_updated \leftarrow$ **false**
/* Local search phase */
6: $best\_set_0 \leftarrow S$, $best\_cost \leftarrow \sum_{v \in S} c_v$, $best\_num\_set \leftarrow 1$
7: **for** all $r \in A_{\mathrm{S}} \setminus S$ **do**
8: $Z \leftarrow \emptyset$
9: **for** all $t \in S$ **do**
10: $Z \leftarrow Z \cup \{t\}$, $H \leftarrow (T \cup S \cup \{r\} \setminus Z, E\downarrow_{T \cup S \cup \{r\} \setminus Z})$
11: Calculate $num\_uncovered$ in H using $Distance_G$ and $l_{\max}$
12: **if** $num\_uncovered > 0$ **then**
13: $Z \leftarrow Z \setminus \{t\}$
14: **end if**
15: **end for**
16: **if** $\sum_{v \in S \cup \{r\} \setminus Z} c_v < best\_cost$ **then**
17: $best\_num\_set \leftarrow 0$
18: **end if**
19: **if** $\sum_{v \in S \cup \{r\} \setminus Z} c_v \leq best\_cost$ **and** $S \cup \{r\} \setminus Z \notin best\_set$ **then**
20: $best\_set_{best\_num\_set} \leftarrow S \cup \{r\} \setminus Z$,
$best\_cost \leftarrow \sum_{v \in S \cup \{r\} \setminus Z} c_v$,
$best\_num\_set \leftarrow best\_num\_set + 1$
21: **end if**
22: **end for**
23: **if** $best\_cost < \sum_{v \in S} c_v$ **then**
24: $S \leftarrow$ select a set randomly from $best\_set$
25: $solution\_updated \leftarrow$ **true**
26: **end if**
27: **while** $solution\_updated$
/* Best solution update */
28: **if** $\sum_{v \in S} c_v < best\_value$ **then**
29: $S^* \leftarrow S$, $best\_value \leftarrow \sum_{v \in S} c_v$
30: **end if**
31: **end for**
32: **return** S^*

the cost function c, the pre-computed $Distance_G$ table, the maximum acceptable path length $l_{\max}$, and the number of iterations ($max\_iterations$). In each iteration, the construction phase to find the initial set of sinks S is executed in line 3. The local search starts with the initialisation of the best set and the best cost in line 6. The loop from line 7 to 22 searches for the best move, i.e. finding a new sink $r \in A_S \setminus S$ that can eliminate as many existing sinks from S as possible. The loop from line 9 to 15 tries to find the set $Z \subseteq S$ of existing sinks that are safe to be removed after the insertion of r. The sinks in Z are safe to be removed if all sensors in $H = (T \cup S \cup \{r\} \setminus Z, E{\downarrow_{T \cup S \cup \{r\} \setminus Z}})$ are double-covered. In line 16, we check if the new solution reduces the total cost of the current best solution. If the total cost can be reduced, we reset the set of the best set in line 17. If the total cost is the same, we keep this new solution in the set of the best set as shown from line 19 to 21. When all moves have been evaluated, we check in line 23 if an improving solution has been found. If the moves produce a better solution, the set of sinks S is updated in line 24 by selecting one best set randomly from the set of the best set. Then, the local search continues. If, at the end of the local search, we find a better solution compared to the best solution found so far, we update in line 29 the set of sinks and the lowest total cost found. The best sink set S^* is returned in line 32.

7.3 Evaluation of Greedy-MSP and GRASP-MSP

We evaluate Greedy-MSP and GRASP-MSP, and we show that while Greedy-MSP has the shortest runtime, GRASP-MSP finds the lowest cost sink deployment compared to other closely-related algorithms. GRASP-MSP finds comparable solution to the optimal with faster runtime than the optimal. We measure the performance of the algorithms using the following metrics:

1. ***Number of sinks needed and total sink cost***. We compare the effectiveness of the algorithms in finding the minimum cost solution to the same problem. We expect that GRASP-MSP deploys the fewest sinks and has the

lowest cost solution compared to other algorithms. We also expect Greedy-MSP's solution to have slightly more sinks than GRASP-MSP's, but it should be comparable to other multiple sink placement algorithms.

2. ***Runtime***. We expect that Greedy-MSP and GRASP-MSP are efficient as the computation time is faster compared to other algorithms.

The results presented here are based on the mean value of 20 randomly generated network deployments. The network consists of 100 sensor nodes deployed within randomly perturbed grids, where a sensor node is placed in a unit grid square of 8m $\times$ 8m and the coordinates are perturbed. To get sparse networks (average degree 2-3), we generate more grid points than the number of nodes. We use 11×11 grid squares to randomly deploy 100 sensor nodes. 25 candidate sinks are also distributed in a grid area, where a candidate occupies a unit grid square of 18m $\times$ 18m. Both sensor nodes and sinks use 10-metre transmission range.

We compare Greedy-MSP and GRASP-MSP to *Minimise the Number of Sinks for Fault-Tolerance* (MSFT), *Cluster-Based Sampling for Multiple Sink Placement* (CBS-MSP) and the optimal solution. Since there are no existing algorithms in the literature that share the same objectives as ours, we take the closest approaches and modify them to be comparable. MSFT and CBS-MSP are algorithms based on the well-known k-means clustering algorithm. MSFT is similar to *Minimise the Number of Sinks for a Predefined Minimum Operation Period* (MSPOP) [85]. In MSPOP, sinks can be deployed anywhere and they are placed one by one until a required lifetime is met. Unlike MSPOP, MSFT has candidate locations and keeps adding sinks until the network is double-covered. CBS-MSP is similar to *Cluster-Based Sampling* (CBS) proposed in [28], but with some modifications. In CBS, the number of sinks is given and the objective is to minimise the total road distance from all nodes to the sinks where each node is required to be double-covered. Unlike CBS, CBS-MSP tries to reduce the number of sinks and thus the deployment cost. We implement CBS-MSP using path length to represent distance between two nodes

and also we have path length restrictions. In each iteration, both CBS and CBS-MSP try to find the best sink locations to ensure the network is double-covered. The k-means clustering algorithm is used in these algorithms to divide the network into clusters and to find the position of each sink, which is in the centre of a cluster. The pseudocode for MSFT and CBS-MSP are given in Appendix D.1.

The performances of MSFT and CBS-MSP depend on the randomly selected sink locations. Therefore, we use the maximum iteration (*MaxIter*) to limit the number of iterations. We vary the number of iterations for MSFT and CBS-MSP, while we use $MaxIter = 1$ and 10 for GRASP-MSP as has been shown in the previous chapters that GRASP-based algorithms with $MaxIter = 10$ produce similar results to higher number of iterations. In our Greedy-MSP and GRASP-MSP, if two or more moves offer the same solution, we select one arbitrarily. We also consider Greedy-MSP-All and GRASP-MSP-All, where if there exists more than one best move, we evaluate them all.

The optimal solution is modeled using binary linear programming with the objective is to minimise the total sink cost, i.e.

$$\min \sum_{j \in S} c_j x_j \tag{7}$$

subject to the following constraints

$$\sum_{j \in S} l_{ij} x_j \geq 2; \ \forall i \in T \tag{8}$$

$$d_{ij} \leq l_{\max} \Rightarrow l_{ij} = 1, \ d_{ij} > l_{\max} \Rightarrow l_{ij} = 0; \ i \in T, \ j \in S \tag{9}$$

$$x_j \in \{0, 1\}; \ j \in S \tag{10}$$

c is the cost of a candidate sink x. The first constraint guarantees each sensor node has at least two paths to two sinks. The paths are length-bounded, which are shown in the second constraint using a binary function l_{ij}. It has value equal to one if the shortest path length from a sensor node i to a sink $j \leq l_{\max}$, otherwise its value is zero. In the third constraint, a candidate sink is either selected to be deployed

or not. The binary linear programming for the optimal solution is implemented in Matlab, while the other algorithms are written in C++.

In the simulation, we find the best locations to deploy the least number of sinks to make the networks double-covered. We assume all candidate sinks have the same cost and consider the cases where the maximum acceptable path length from each sensor node to a sink is 6 and 10. The number of sinks deployed by each algorithm is shown in Figure 82 and the runtime is in Table 17. The simulation results show that GRASP-MSP with $MaxIter = 1$ requires almost the same number of sinks with shorter runtime compared to the optimal solution. It also outperforms MSFT, CBS-MSP and Greedy-MSP. Greedy-MSP has the shortest runtime, but it places more sinks compared to GRASP-MSP. We also observe that the number of deployed sinks decreases when the maximum path length increases because each sink in the network can cover more sensor nodes.

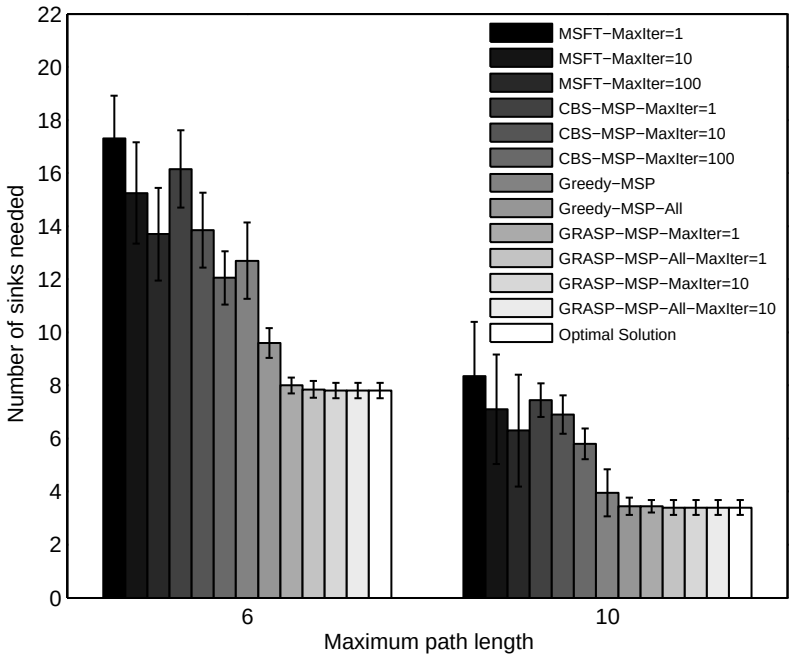


Figure 82: Number of sinks needed for multiple sink placement algorithms versus maximum path length

Table 17: Multiple sink placement algorithms' runtime with different maximum path length

Algorithms	**Runtime (sec)**	
	$l_{\max}=6$	$l_{\max}=10$
MSFT-MaxIter=1	0.4188	0.0288
MSFT-MaxIter=10	4.0429	0.3102
MSFT-MaxIter=100	40.8313	3.1647
CBS-MSP-MaxIter=1	1.0297	0.0235
CBS-MSP-MaxIter=10	10.1797	0.2069
CBS-MSP-MaxIter=100	97.0899	2.0844
Greedy-MSP	0.0024	0.0040
Greedy-MSP-All	7.9664	0.0071
GRASP-MSP-MaxIter=1	0.0085	0.0117
GRASP-MSP-All-MaxIter=1	0.0179	0.0116
GRASP-MSP-MaxIter=10	0.0688	0.0452
GRASP-MSP-All-MaxIter=10	0.1305	0.0750
Optimal Solution	0.0727	0.0867

We also consider a more realistic scenario where the deployment costs for all candidate sinks are different from one another, for example due to cabling and installation costs. We do not specify the locations where the costs are higher or lower, but we assume a set of candidate locations with their associated costs. We compare the performance of the algorithms using the same sink cost (c_{S}), i.e. 3 units and different sink costs, which are randomly selected between 3 and 6 units. The results for the total sink cost and the runtime with $l_{\max}=6$ are presented in Figure 83 and Table 18, respectively. These results show similar trends to the previous ones when we vary the maximum path lengths. That is, GRASP-MSP achieves the lowest total cost comparable to the optimal solution, while Greedy-MSP has the shortest runtime in all scenarios.

We evaluate the performance of GRASP-MSP against the optimal solution when the density of the network increases. In the simulation, we double the number of sensor nodes from 100 to 200 while keeping the area fixed. As a result, the average degree of a sensor node increases from 3 to 7. We present the number of deployed sinks out of 25 candidate sinks in Figure 84 and the runtime in Table 19. GRASP-MSP with $MaxIter = 1$ has the shortest runtime, but it places slightly more sinks than

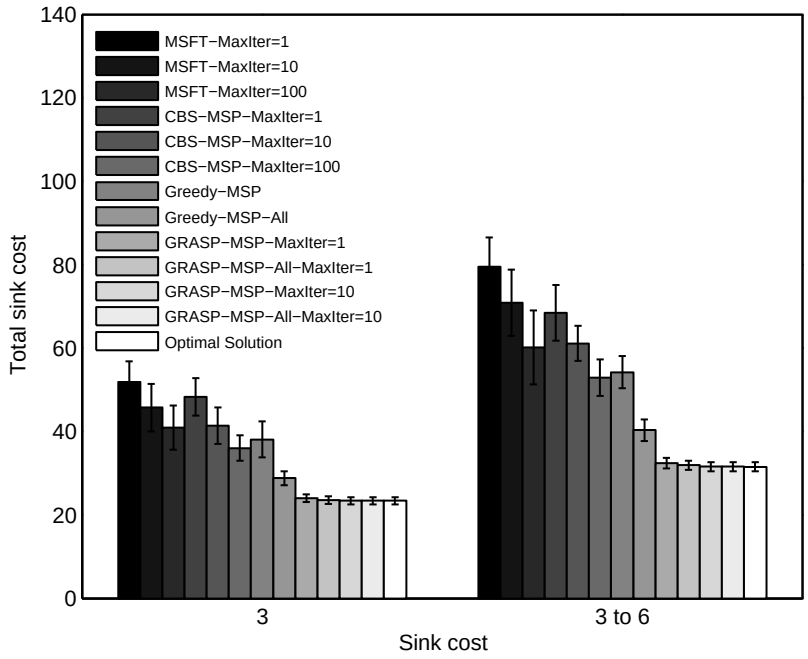


Figure 83: Total sink cost for multiple sink placement algorithms versus sink cost

Table 18: Multiple sink placement algorithms' runtime with different sink cost

Algorithms	**Runtime (sec)**	
	$c_S = 3$	$c_S = 3$ **to** 6
MSFT-MaxIter=1	0.4188	0.4086
MSFT-MaxIter=10	4.0429	4.1327
MSFT-MaxIter=100	40.8313	42.3492
CBS-MSP-MaxIter=1	1.0297	0.9258
CBS-MSP-MaxIter=10	10.1797	9.9290
CBS-MSP-MaxIter=100	97.0899	96.5509
Greedy-MSP	0.0024	0.0008
Greedy-MSP-All	7.9664	8.1971
GRASP-MSP-MaxIter=1	0.0085	0.0118
GRASP-MSP-All-MaxIter=1	0.0179	0.0126
GRASP-MSP-MaxIter=10	0.0688	0.1187
GRASP-MSP-All-MaxIter=10	0.1305	0.1335
Optimal Solution	0.0727	0.1086

Table 19: Multiple sink placement algorithms' runtime with different average degree

Algorithms	**Runtime (sec)**	
	average degree = 3	**average degree = 7**
GRASP-MSP-MaxIter=1	0.0085	0.0093
GRASP-MSP-All-MaxIter=1	0.0179	0.0337
GRASP-MSP-MaxIter=10	0.0688	0.1102
GRASP-MSP-All-MaxIter=10	0.1305	0.2821
Optimal Solution	0.0727	0.0735

the optimal solution. Increasing the number of GRASP-MSP's iterations results in longer runtime but it has the same solution as the optimal.

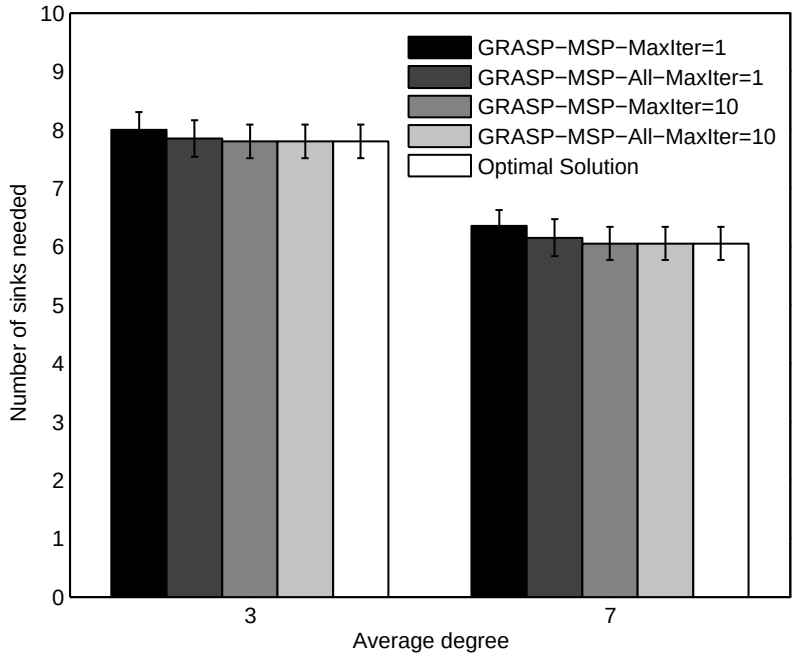


Figure 84: Number of sinks needed for GRASP-MSP and the optimal solution versus average degree

At this stage, finding the optimal solution is sufficient for the multiple sink placement problem. Nevertheless, the GRASP-MSP performance gives us confidence to use the same local search technique for the more complex multiple sink and relay placement problem, where a linear optimal solution is not available.

7.4 Multiple Sink and Relay Placement (MSRP)

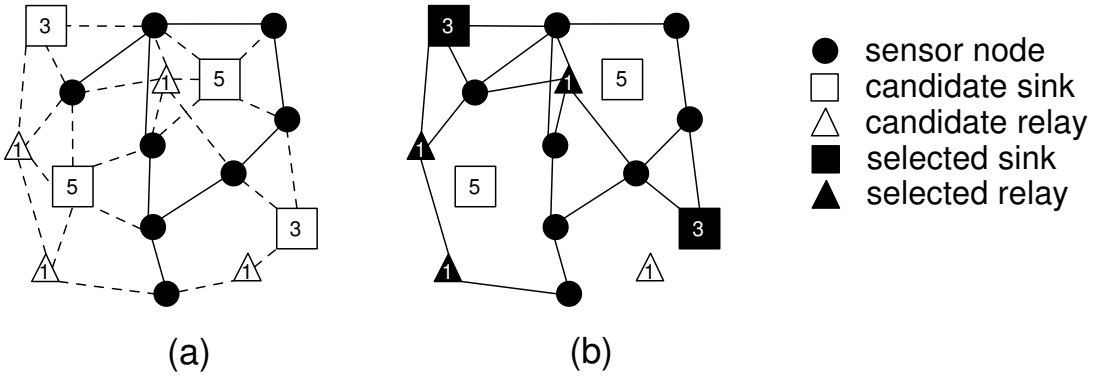


Figure 85: Illustration of the MSRP problem. (a) A WSN with four candidate sinks and four candidate relays, and (b) the double-covered and noncritical WSN where $l_{\max}=3$.

For the sink and relay placement, vertices are partitioned into a set of sensors T, relays R and sinks S. In the graph representation, $V=T\cup R\cup S$. We identify a sensor as *critical* if and only if upon its removal, more sensors will have no path of length $\leq l_{\max}$ to a sink. Otherwise, it is *non-critical*. We define a WSN as non-critical if each sensor $v\in T$ is non-critical. In the *multiple sink and relay placement problem*, given a graph $G=(T\cup A_{\mathrm{R}}\cup A_{\mathrm{S}},E)$, where A_{R} and A_{S} are sets of candidate locations for relays and sinks, respectively, we find minimum cost subsets $R\subseteq A_{\mathrm{R}}$ and $S\subseteq A_{\mathrm{S}}$ such that $H=(T\cup R\cup S,E{\downarrow_{T\cup R\cup S}})$ is double-covered and non-critical. The relay and sink candidate locations are associated with a non-negative cost function $c:A_{\mathrm{R}}\cup A_{\mathrm{S}}\rightarrow\mathbb{R}$. We assume that a relay is cheaper than a sink because sinks usually are assumed to be powered, have more memory, processing and WiFi/ethernet backhaul.

The multiple sink and relay placement problem is illustrated in Figure 85. Figure 85(a) shows a WSN with four candidate sinks and four candidate relays. The numbers in the figure represent the costs. In this example, if we choose the two 5-unit sinks, we only need to deploy the relay at the bottom-left corner to make the network double-covered and non-critical. The total cost is 11 units. However, if we choose the two 3-unit sinks as shown in Figure 85(b), we need three additional

relays. The total cost of this deployment is the lowest, i.e. 9 units.

In this section, we present Greedy-MSRP and GRASP-MSRP to solve the multiple sink and relay placement problem. Both algorithms use the concept of Length-constrained Connectivity and Rerouting Centrality (l-CRC) from Chapter 6 to identify critical sensors. A sensor is critical if its centrality index is above a given threshold. We can raise the threshold to trade-off the deployment cost against the robustness of the network. However, in this chapter, we only assume zero threshold for full reliability. After the identification of critical sensors, some candidate relays are selected to be deployed. We identify relays that need to be deployed by finding the shortest path from each descendant of each critical sensor to a sink bypassing each critical sensor.

We will firstly present Greedy-MSRP. Greedy-MSRP uses Greedy-MSP to deploy a minimum number of sinks and GRASP-MRP to deploy a minimum number of relays. Before presenting Greedy-MSRP, we will look at GRASP-MRP, which is a modification of GRASP-ABP from Chapter 6 to tackle both double-covered and non-critical requirements.

7.4.1 Greedy Randomised Adaptive Search Procedure for Multiple Relay Placement (GRASP-MRP)

GRASP-ABP in Chapter 6 only deploys relays to create non-critical networks. As we consider both double-covered and non-critical requirements in this chapter, we present GRASP-MRP for multiple relay placement.

Construction Phase

In the construction phase, we find $R \subseteq A_{\mathrm{R}}$, an initial set of relays from the candidate relays, to minimise the number of uncovered and critical sensors. Initially R is an empty set. Given sets of sensors T and sinks S, we firstly find and store the

length of the shortest path of each sensor to the sinks in $Distance_H$ table, where $H = (T \cup R \cup S, E\downarrow_{T \cup R \cup S})$. If a sensor $v \in T$ is uncovered, we try to place some relays to construct a path to a sink $w \in S$ if $Distance_H(v, w) > l_{\max}$ but the pre-computed $Distance_G(v, w) \leq l_{\max}$. We choose the relays that appear on the shortest path from v to w by tracing the path in $Parent_G(v, w)$. If a sensor needs two paths to make it double-covered, this step is repeated twice. After some relays are selected from the candidate relays, we have the set of relays R that minimises the number of uncovered sensors. We then rebuild $H = (T \cup R \cup S, E\downarrow_{T \cup R \cup S})$ and check if critical sensors exist using centrality calculation. If a sensor $v \in T$ is critical, we deploy relays that appear on the shortest path from each descendant of v to a sink $w \in S$ bypassing v, as long as the shortest path length is $\leq l_{\max}$. The randomisation of the initial solution is obtained by randomly selecting parents in the shortest paths if there are hop count ties.

Node-based Local Search

Let R be the set of relays. The local search's move tries to add a new relay $r \in A_{\mathrm{R}} \backslash R$ into R that can eliminate as many existing relays from R as possible. This move must always ensure that the network is double-covered and non-critical.

Algorithm Description

Algorithm 14 shows the pseudocode for GRASP-MRP. It takes as input the original graph $G = (T \cup A_{\mathrm{R}} \cup A_{\mathrm{S}}, E)$, the set T of sensors, the set S of sinks, the set A_{R} of candidate relays, the maximum acceptable path length $l_{\max}$, and the number of iterations (*max_iterations*). In each iteration, the construction phase to find the initial set of relays R to minimise the number of uncovered and critical sensors is executed in line 3. The local search starts with the initialisation of the best set and the best number of relays in line 6. The loop from line 7 to 20 searches for the best move to find a new relay $r \in A_{\mathrm{R}} \backslash R$ that can eliminate as many existing relays

Algorithm 14: GRASP-MRP

Input : $G, T, S, A_{\mathrm{R}}, l_{\max}, max\_iterations$
Output: R^*

1: $best\_value \leftarrow \infty$
2: **for** $i \leftarrow 1$ to $max\_iterations$ **do**
/* Construction phase */
3: Find initial R
4: **do**
5: $solution\_updated \leftarrow$ **false**
/* Local search phase */
6: $best\_set \leftarrow R, \quad best\_number \leftarrow |R|$
7: **for** all $r \in A_{\mathrm{R}} \setminus R$ **do**
8: $Z \leftarrow \emptyset$
9: **for** all $t \in R$ **do**
10: $Z \leftarrow Z \cup \{t\}, \quad H \leftarrow (T \cup R \cup \{r\} \setminus Z \cup S, E{\downarrow_{T \cup R \cup \{r\} \setminus Z \cup S}})$
11: Calculate $Distance_H$
12: Calculate $num\_uncovered$ and $num\_critical$ in H using $Distance_H$ and $l_{\max}$
13: **if** $num\_uncovered > 0$ **or** $num\_critical > 0$ **then**
14: $Z \leftarrow Z \setminus \{t\}$
15: **end if**
16: **end for**
17: **if** $|R| - |Z| + 1 < best\_number$ **then**
18: $best\_set \leftarrow R \cup \{r\} \setminus Z, \quad best\_number \leftarrow |R| - |Z| + 1$
19: **end if**
20: **end for**
21: **if** $best\_number < |R|$ **then**
22: $R \leftarrow best\_set$
23: $solution\_updated \leftarrow$ **true**
24: **end if**
25: **while** $solution\_updated$
/* Best solution update */
26: **if** $|R| < best\_value$ **then**
27: $R^* \leftarrow R, \quad best\_value \leftarrow |R|$
28: **end if**
29: **end for**
30: **return** R^*

from R as possible. The loop from line 9 to 16 finds the set $Z \subseteq R$ of existing relays that are safe to be removed after the insertion of r. The relays in Z are safe to be removed if all sensor in $H = (T \cup R \cup \{r\} \setminus Z \cup S, E\downarrow_{T \cup R \cup \{r\} \setminus Z \cup S})$ are double-covered and non-critical. We check in line 17 if the new solution has fewer relays than the current best solution. If the number of relays can be reduced, the best set and the best number of relays are updated in line 18. When all local search moves have been evaluated, we check if an improving solution has been found in line 21. If the moves produce a better solution, the set of relays R is updated in line 22. Then, the local search continues. If, at the end of the local search, we find a better solution compared to the best solution found so far, we update in line 27 the set of relays and the least number of relays used. Finally, the best relay set R^* is returned in line 30.

7.4.2 Greedy Algorithm for Multiple Sink and Relay Placement (Greedy-MSRP)

After describing GRASP-MRP, we are now ready to present Greedy-MSRP for the multiple sink and relay placement problem. Greedy-MSRP uses Greedy-MSP to select a minimum number of sinks to make a network double-covered and GRASP-MRP to deploy a minimum number of relays to make the network non-critical. Since the cost of a sink is assumed to be more expensive than the cost of a relay, Greedy-MSRP tries to reduce the total deployment cost by trading some sinks with relays. The deployed sinks are removed one by one and more relays are added in the network. However, this swap must ensure that the network is always double-covered and non-critical.

The Greedy-MSRP pseudocode is given in Algorithm 15. It takes as input the original graph $G = (T \cup A_{\mathrm{R}} \cup A_{\mathrm{S}}, E)$, the set T of sensors, the set A_{R} of candidate relays, the set A_{S} of candidate sinks, the cost function c, the pre-computed $Distance_G$ table, the maximum acceptable path length $l_{\max}$, and the number of iterations

Algorithm 15: Greedy-MSRP

Input : $G, T, A_{\mathrm{R}}, A_{\mathrm{S}}, c, Distance_G, l_{\max}, max\_iterations$
Output: R^*, S^*

1: $best\_cost \leftarrow \infty$
2: $S \leftarrow$ Greedy-MSP$(G, T, A_{\mathrm{S}}, c,\ Distance_G, l_{\max})$
3: $num\_sink \leftarrow |S|$
4: **for** $n \leftarrow num\_sink$ downto 2 **do**
5: $H \leftarrow (T \cup A_{\mathrm{R}} \cup S, E{\downarrow_{T \cup A_{\mathrm{R}} \cup S}})$
6: Calculate $num\_uncovered$ and $num\_critical$ in H using $Distance_G$ and $l_{\max}$
7: **if** $num\_uncovered = 0$ **and** $num\_critical = 0$ **then**
8: $R \leftarrow$ GRASP-MRP$(G, T, S, A_{\mathrm{R}}, l_{\max},\ max_iterations)$
9: **if** $\sum_{v \in R \cup S} c_v < best\_cost$ **then**
10: $R^* \leftarrow R, \quad S^* \leftarrow S, \quad best\_cost \leftarrow \sum_{v \in R \cup S} c_v$
11: **end if**
12: **end if**
13: $S \leftarrow S \setminus \{Sink_n\}$
14: **end for**
15: **return** R^*, S^*

($max\_iterations$) for GRASP-MRP. The best cost is initialised in line 1 and Greedy-MSP is called in line 2 to find the set of sinks S. Greedy-MSRP iterates from line 4 to 14 to deploy relays. In the first iteration, all sinks in S are placed. Then, the sinks are gradually removed, starting from the last one inserted into S. We check the solution in line 7 if all sensors are double-covered and non-critical in $H = (T \cup A_{\mathrm{R}} \cup S, E{\downarrow_{T \cup A_{\mathrm{R}} \cup S}})$. If this is the case, GRASP-MRP is called in line 8 to find the set of relays R. If the new total cost of sinks and relays is less than the best cost found so far, the best set of relays R^*, the best set of sinks S^* and the best cost are updated in line 10. R^* and S^* are returned in line 15.

7.4.3 Greedy Randomised Adaptive Search Procedure for Multiple Sink and Relay Placement (GRASP-MSRP)

We now present GRASP-MSRP to solve the multiple sink and relay placement problem. Unlike Greedy-MSRP that deploys relays after finding the minimal set of sinks, GRASP-MSRP finds the least deployment cost by placing sinks and relays at the same time. We give the detailed algorithm below.

Construction Phase

In the construction phase, we find $R \subseteq A_{\mathrm{R}}$ and $S \subseteq A_{\mathrm{S}}$ as our initial sets of relays and sinks, respectively. Initially R and S are empty sets. We then alternate the sink and relay addition during this process. We deploy some sinks to minimise the number of uncovered sensors and then some relays to minimise the number of both uncovered and critical sensors. Note that we do not add more sinks if at some points the network is already double-covered. After the addition of either sinks or relays, $H = (T \cup R \cup S, E{\downarrow_{T \cup R \cup S}})$ is rebuilt. The process of sink and relay addition is repeated until the network is double-covered and non-critical.

We need at least two sinks for a double-covered WSN, so we firstly choose two sinks randomly from A_{S}. We then deploy a bunch of relays from A_{R} to minimise the number of uncovered and critical sensors. If a sensor $v \in T$ is uncovered, we place some relays to construct a path to a sink $w \in S$ if $\mathit{Distance}_H(v, w) > l_{\max}$ but $\mathit{Distance}_G(v, w) \leq l_{\max}$. We choose the relays that appear on the shortest path from v to w by tracing the path in $\mathit{Parent}_G(v, w)$. If the sensor needs two paths to make it double-covered, we repeat this step twice. If a sensor $v \in T$ is critical, we deploy relays that appear on the shortest path from each descendant of v to a sink $w \in S$ bypassing v, as long as the shortest path length is $\leq l_{\max}$. After the relay deployment, we place sinks again. In order to add the randomisation to the initial solution, we randomly select parents in the shortest paths if there are hop count ties.

Node-based Local Search

Let R be the set of relays and S be the set of sinks. We look for a lower cost solution by adding either a new relay $r \in A_{\mathrm{R}} \setminus R$ into R or a new sink $s \in A_{\mathrm{S}} \setminus S$ into S that can eliminate some existing relays from R and sinks from S to minimise the total cost as possible. Given that the cost of a sink is higher than the cost of a relay, we also try to minimise the total cost by adding some relays into R when we eliminate

an existing sink from S. The local search moves are performed to reduce the total cost, but must ensure that the network is always double-covered and non-critical in each iteration.

Algorithm Description

The GRASP-MSRP pseudocode is given in Algorithm 16. Generally, its concept is similar to GRASP-MSP in Algorithm 13 with some key differences. The key differences are the inclusion of candidate relays as one of its input, the identification of critical sensors, the deployment of relays to minimise the number of uncovered and critical sensors, and the repetitive computation of the shortest path from all sensors to all sinks due to the addition and elimination of relays. The detailed description of the pseudocode is given below.

GRASP-MSRP takes as input the original graph $G = (T \cup A_{\mathrm{R}} \cup A_{\mathrm{S}}, E)$, the set T of sensors, the set A_{R} of candidate relays, the set A_{S} of candidate sinks, the cost function c, the maximum acceptable path length $l_{\max}$, and the number of iterations (*max_iterations*). In each iteration, the construction phase to find initial sets of relays R and sinks S is executed in line 3. The local search starts with the initialisation of the best set and the best cost in line 6. The loop from line 7 to 39 searches for the best move, i.e. finding either a new relay or a new sink $r \in A_{\mathrm{R}} \cup A_{\mathrm{S}} \setminus W$ that can eliminate as many existing relays and sinks from W as possible, where $W = R \cup S$. The loop from line 9 to 32 tries to find the set $Z \subseteq W \cup X$ of existing relays and sinks that are safe to be removed after the insertion of Y. Y is the set of new relays and sinks that are added during the iteration. X is the set of new relays that are added to the network to reduce the total cost when a sink is removed from the network.

The algorithm checks for uncovered sensors in line 13. If some exist, it tries to deploy some relays in line 15. The identification of critical sensors is performed in line 19. If some exist, relays are added in line 21. Note that we try to minimise the total cost by adding some relays when we eliminate a sink. These relays are

Algorithm 16: GRASP-MSRP

Input : $G, T, A_{\mathrm{R}}, A_{\mathrm{S}}, c, l_{\max}, max\_iterations$
Output: R^*, S^*

1: $best\_value \leftarrow \infty$
2: **for** $i \leftarrow 1$ to $max\_iterations$ **do**
/* Construction phase */
3: Find initial R and S, $W \leftarrow R \cup S$
4: **do**
5: $solution\_updated \leftarrow$ **false**
/* Local search phase */
6: $best\_set_0 \leftarrow W$, $best\_cost \leftarrow \sum_{v \in W} c_v$, $best\_num\_set \leftarrow 1$
7: **for** all $r \in A_{\mathrm{R}} \cup A_{\mathrm{S}} \setminus W$ **do**
8: $Y \leftarrow \{r\}$, $Z \leftarrow \emptyset$, $X \leftarrow \emptyset$
9: **for** all $t \in W \cup X$ **do**
10: $Z \leftarrow Z \cup \{t\}$, $X \leftarrow \emptyset$
11: $H \leftarrow (T \cup W \cup Y \setminus Z, E{\downarrow}_{T \cup W \cup Y \setminus Z})$
12: Calculate $Distance_H$
13: Find $uncovered\_set$ in H using $Distance_H$ and $l_{\max}$
14: **if** $|uncovered\_set| > 0$ **then**
15: $X \leftarrow X \cup \{$Find relays to minimise $|uncovered\_set|\}$
16: **end if**
17: $H \leftarrow (T \cup W \cup X \cup Y \setminus Z, E{\downarrow}_{T \cup W \cup X \cup Y \setminus Z})$
18: Calculate $Distance_H$
19: Find $critical\_set$ in H using $Distance_H$ and $l_{\max}$
20: **if** $|critical\_set| > 0$ **then**
21: $X \leftarrow X \cup \{$Find relays to minimise $|critical\_set|\}$
22: **end if**
23: $H \leftarrow (T \cup W \cup X \cup Y \setminus Z, E{\downarrow}_{T \cup W \cup X \cup Y \setminus Z})$
24: Calculate $Distance_H$
25: Calculate $num\_uncovered$ and $num\_critical$ in H using $Distance_H$ and $l_{\max}$
26: **if** $num\_uncovered = 0$ **and** $num\_critical = 0$ **then**
27: $Y \leftarrow Y \cup X$, $Z \leftarrow Z \setminus X$
28: **end if**
29: **if** $num\_uncovered > 0$ **or** $num\_critical > 0$ **then**
30: $Z \leftarrow Z \setminus \{t\}$
31: **end if**
32: **end for**
33: **if** $\sum_{v \in W \cup Y \setminus Z} c_v < best\_cost$ **then**
34: $best\_num\_set \leftarrow 0$
35: **end if**
36: **if** $\sum_{v \in W \cup Y \setminus Z} c_v \leq best\_cost$ **and** $W \cup Y \setminus Z \notin best\_set$ **then**
37: $best\_set_{best\_num\_set} \leftarrow W \cup Y \setminus Z$, $best\_cost \leftarrow \sum_{v \in W \cup Y \setminus Z} c_v$
$best\_num\_set \leftarrow best\_num\_set + 1$
38: **end if**
39: **end for**
40: **if** $best\_cost < \sum_{v \in W} c_v$ **then**
41: $W \leftarrow$ select a set randomly from $best\_set$
42: $solution\_updated \leftarrow$ **true**
43: **end if**
44: **while** $solution\_updated$
/* Best solution update */
45: **if** $\sum_{v \in W} c_v < best\_value$ **then**
46: $R^* \leftarrow \emptyset$, $S^* \leftarrow \emptyset$
47: **for** all $v \in W$ **do**
48: **if** $v \in A_{\mathrm{R}}$ **then**
49: $R^* \leftarrow R^* \cup \{v\}$
50: **else**
51: $S^* \leftarrow S^* \cup \{v\}$
52: **end if**
53: **end for**
54: $best\_value \leftarrow \sum_{v \in W} c_v$
55: **end if**
56: **end for**
57: **return** R^*, S^*

saved in X as shown in line 15 and 21, which later will be included in Y, the set of new relays and sinks to be inserted, if X helps the network become double-covered and non-critical. The network is checked if it is double-covered and non-critical in line 25. Note that there are repetitive computation of the shortest path from all sensors to all sinks in line 12, 18 and 24 due to the addition and elimination of relays. The relays and sinks in Z are safe to be removed if without Z all sensors are double-covered and non-critical. In line 33, we check if the new solution improves the total cost of the current best solution. If the total cost is reduced, we reset the set of the best set in line 34. If the total cost is the same, we keep this new solution in the set of the best set as shown from line 36 to 38. When all moves have been evaluated, we check in line 40 if an improving solution has been found. If the moves produce a better solution, the set of relays and sinks W is updated in line 41 by selecting one best set randomly from the set of the best set. After that, the local search continues.

If, at the end of the local search, we find a better solution compared to the best solution found so far, we update from line 46 to 54 the set of relays, the set of sinks, and the lowest total cost found. The best sets R^* of relays and S^* of sinks are returned in line 57.

7.5 Evaluation of Greedy-MSRP and GRASP-MSRP

We evaluate the performance of Greedy-MSRP and GRASP-MSRP using the following metrics:

1. ***Total sink and relay cost***. We want to compare the total deployment cost that resulted from each algorithm, which includes the cost of sinks and the cost of relays. We expect that GRASP-MSRP has the lowest total cost compared to other algorithms.

2. ***Number of devices***, which is divided into number of sinks and number of relays. We present this metric as we cannot infer how many sinks and relays are deployed from the total cost metric. We expect to see that when the sink cost increases, the number of sinks decreases. This happens because some sinks are traded for relays to reduce the deployment cost.

3. ***Runtime***. We also evaluate the efficiency of the algorithms by comparing the algorithms' runtime.

We follow the same simulation setting as for the evaluation of the multiple sink placement problem in Section 7.3, where each network consists of 100 sensor nodes in grid squares of 8m $\times$ 8m and 25 candidate sinks in grid squares of 18m $\times$ 18m. In addition, we also have 81 candidate relays distributed evenly in grid squares of 10m $\times$ 10m.

We compare Greedy-MSRP and GRASP-MSRP against *Minimise the Number of Sinks and Relays for Fault-Tolerance* (MSRFT) and *Cluster-Based Sampling for Multiple Sink and Relay Placement* (CBS-MSRP). The pseudocode for these two algorithms are given in Appendix D.2. MSRFT and CBS-MSRP extend MSFT and CBS-MSP, respectively, to find the best locations to deploy sinks and GRASP-MRP to deploy relays. These two algorithms start by finding the best locations for two sinks before utilising GRASP-MRP to deploy relays. The number of sinks is gradually increased and GRASP-MRP is used to deploy relays until the network becomes double-covered and non-critical. In the simulation, we only use 100 as the maximum iteration (*MaxIter*) for MSRFT and CBS-MSRP.

We evaluate the total deployment cost of the algorithms by varying the sink costs (c_{S}), i.e. 3, 6, randomly between 3 and 6, and 10 units, while the relay cost is fixed at 1 unit. The total sink and relay cost suggested by each algorithm with $l_{\max} = 6$ is presented in Figure 86 and the runtime is in Table 20. We also show the total numbers of sinks and relays for each algorithm from Figure 87 to Figure 91.

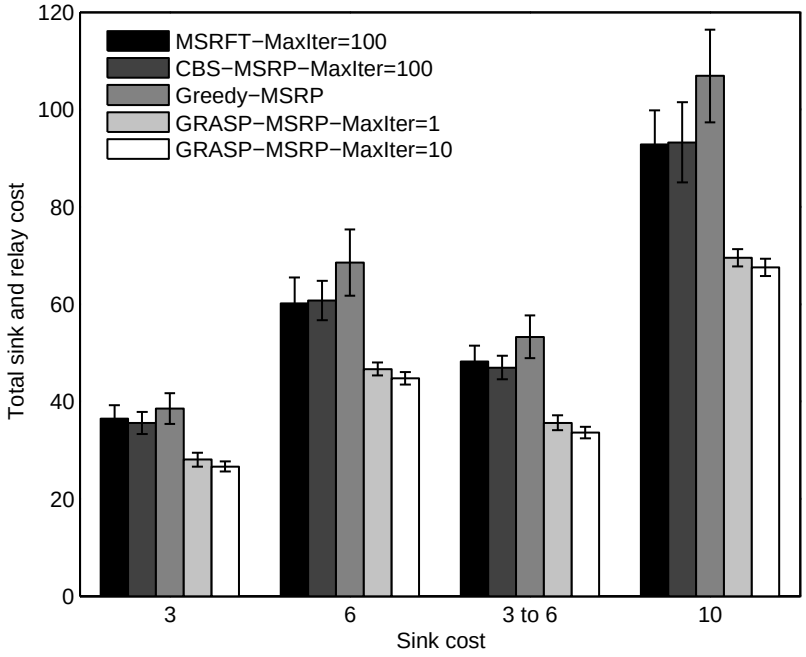


Figure 86: Total sink and relay cost for multiple sink and relay placement algorithms versus sink cost

Table 20: Multiple sink and relay placement algorithms' runtime with different sink cost

Algorithms	**Runtime (sec)**			
	$c_S = 3$	$c_S = 6$	$c_S = 3$ **to** 6	$c_S = 10$
MSRFT-MaxIter=100	61.9235	60.3157	60.1228	59.2399
CBS-MSRP-MaxIter=100	61.2929	64.0727	62.0789	61.9352
Greedy-MSRP	153.7541	146.8796	130.1220	141.8679
GRASP-MSRP-MaxIter=1	20.4173	22.693	23.2985	21.0586
GRASP-MSRP-MaxIter=10	196.5039	216.4508	251.2635	228.3422

The results in Figure 86 show that GRASP-MSRP has the lowest total deployment cost compared to other algorithms. This is because GRASP-MSRP has the fewest sinks if we compare the number of deployed sinks from Figure 87 to Figure 91. The simulation also shows that we are able to trade-off GRASP-MSRP's runtime for a reduced cost when we increase the number of iterations from 1 to 10.

Greedy-MSRP is outperformed by the two k-means clustering-based algorithms, MSRFT and CBS-MSRP. Firstly in terms of the total deployment cost, Greedy-MSRP deploys more sinks than MSRFT and CBS-MSRP as shown from Figure 89 to Figure 91. Secondly for the runtime, Greedy-MSRP is slower because it tries to find the lowest cost solution by solving the multiple sink and relay placement problem from the number of sink = 2 to n, where n is the number of sinks found by Greedy-MSP. On the other hand, MSRFT and CBS-MSRP start from the number of sink = 2 and stop when the network is double-covered and non-critical.

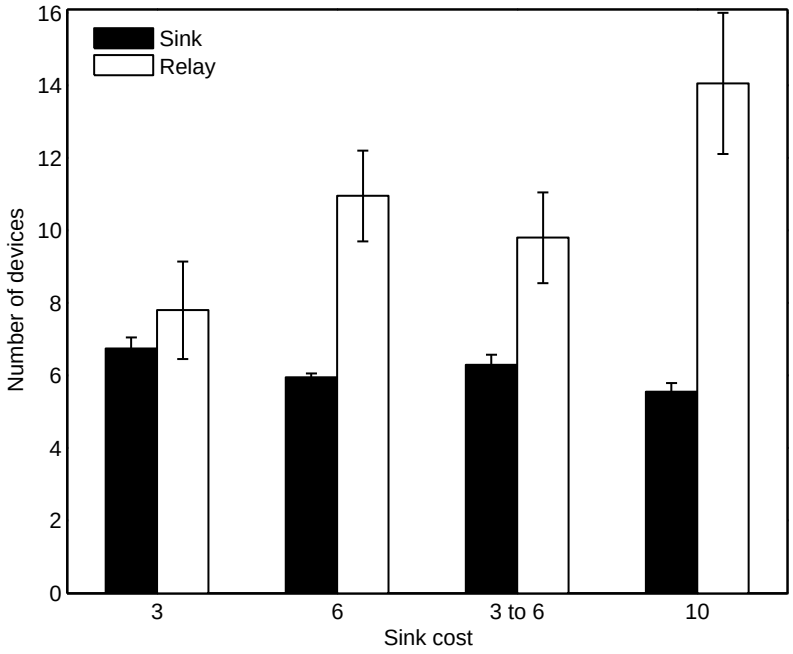


Figure 87: Total numbers of sinks and relays for GRASP-MSRP with $MaxIter = 1$ versus sink cost

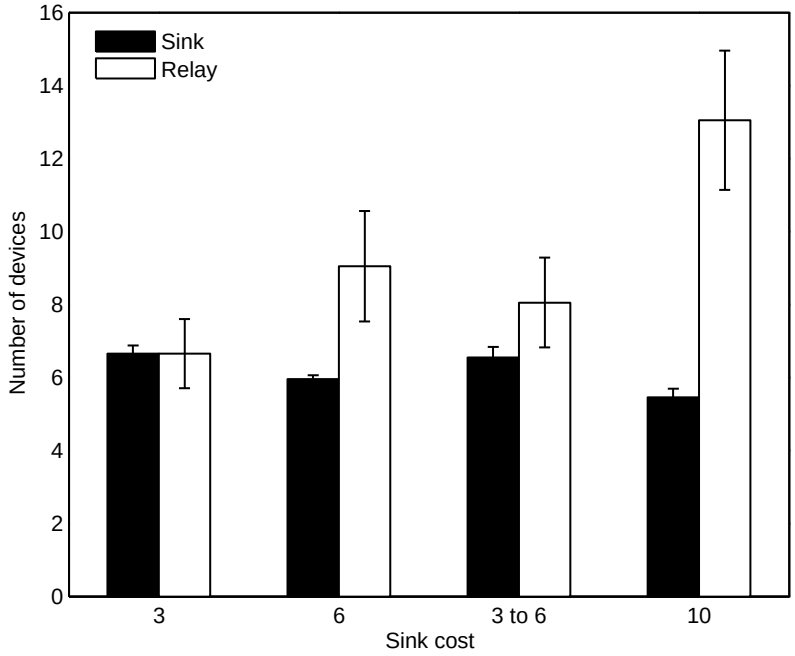


Figure 88: Total numbers of sinks and relays for GRASP-MSRP with $MaxIter = 10$ versus sink cost

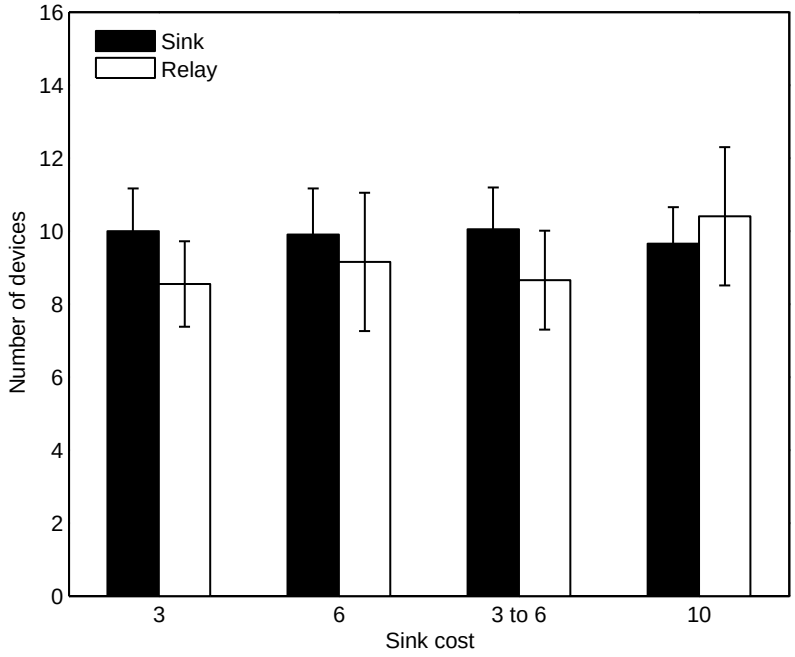


Figure 89: Total numbers of sinks and relays for Greedy-MSRP versus sink cost

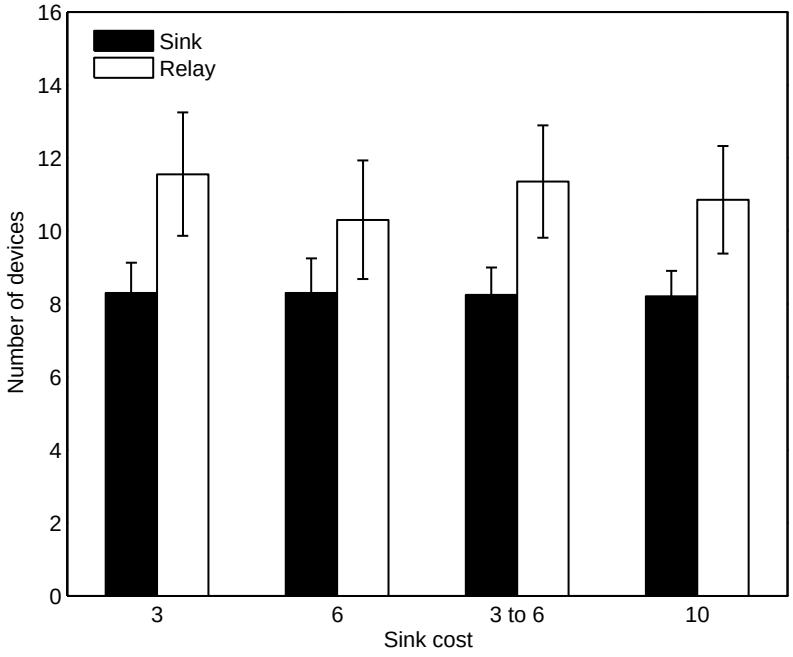


Figure 90: Total numbers of sinks and relays for MSRFT versus sink cost

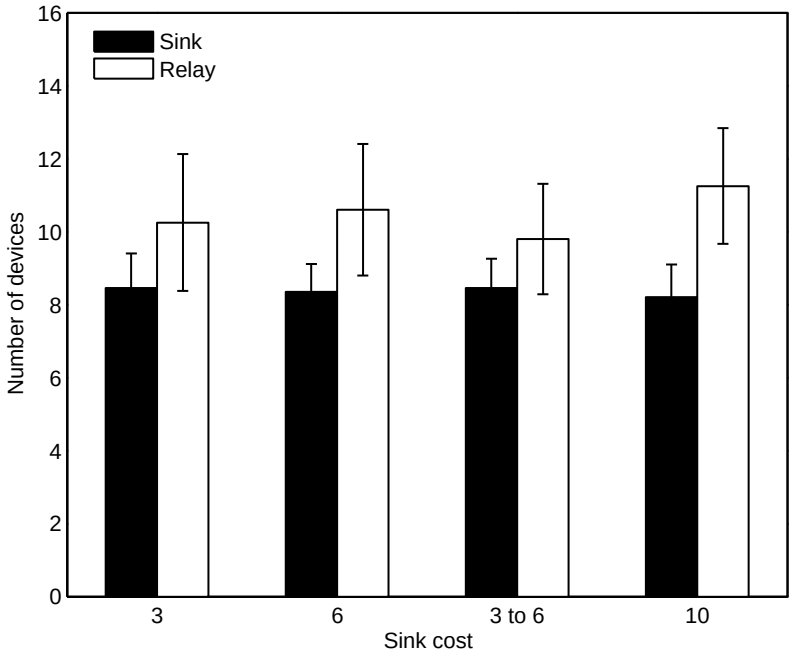


Figure 91: Total numbers of sinks and relays for CBS-MSRP versus sink cost

GRASP-MSRP can swap sinks with relays to reduce the total deployment cost. Therefore, when the sink cost increases, the number of sinks decreases because more sinks are exchanged with relays as shown in Figure 87 and Figure 88. This phenomenon can also be observed in Greedy-MSRP, MSRFT and CBS-MSRP as depicted in Figure 89, Figure 90 and Figure 91, respectively. While the reduction in the number of sinks for Greedy-MSRP is very small when the sinks become more expensive, it is hardly noticeable for MSRFT and CBS-MSRP.

Since GRASP-MSRP gives the best solution for the multiple sink and relay placement problem, we further investigate its performance under various simulation settings by using $MaxIter = 10$. Firstly, we increase $l_{\max}$ from 6 to 10, while keeping the sink cost fixed at 3 units. The results are depicted in Figure 92. When $l_{\max}$ is increased from 6 to 10, the number of required sinks drops significantly from 6.65 to 2.55, while the number of relays does not increase much. The runtime of GRASP-MSRP also increases from 196.5039 seconds to 348.6041 seconds.

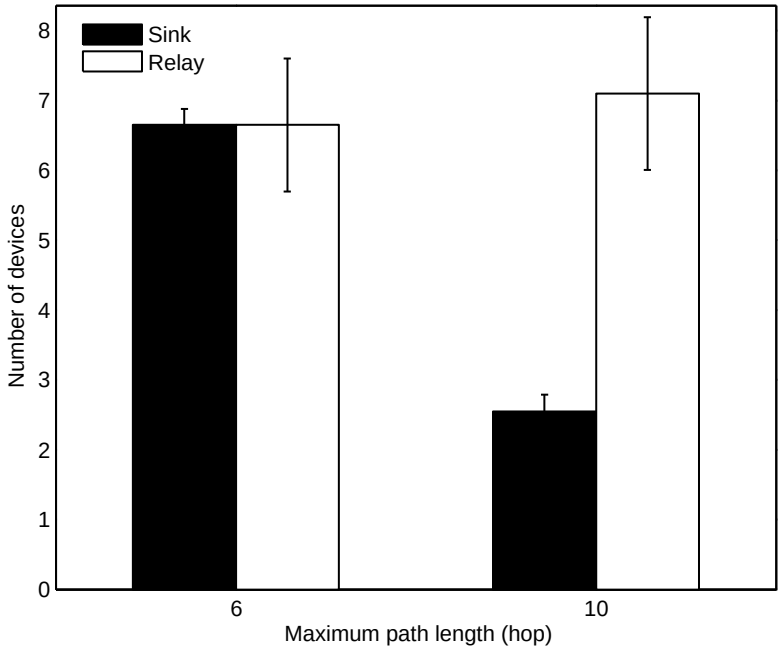


Figure 92: Total numbers of sinks and relays for GRASP-MSRP versus maximum path length

We then try to increase the number of candidate relays from 81, which are evenly distributed in grid squares of 10m × 10m, to 196 candidate relays in grid squares of 6m × 6m. We use a fixed sink cost of 3 units and $l_{\max} = 6$. As shown in Figure 93, when we have more candidate relays, the numbers of deployed sinks and relays slightly decrease because the local search is more likely to find common relays that appear on the shortest paths. The runtime of GRASP-MSRP increases from 196.5039 seconds for 81 candidate relays to 1001.1524 seconds for 196 candidates.

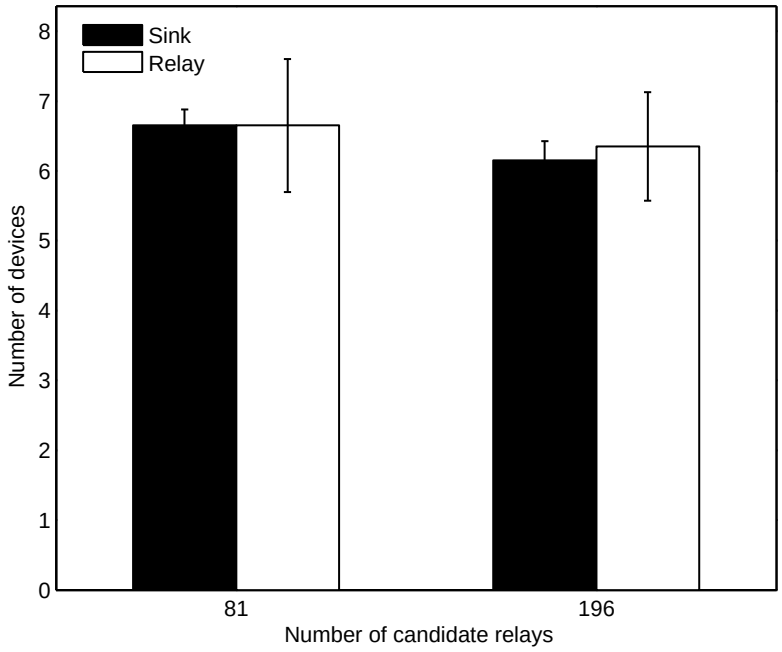


Figure 93: Total numbers of sinks and relays for GRASP-MSRP versus number of candidate relays

We also increase the number of candidate sinks from 25 in grid squares of 18m × 18m to 81 candidates in grid squares of 10m × 10m. The simulation results are presented in Figure 94. When the number of candidate sinks increases, the local search can find better sink positions to cover a network. Therefore, the network requires fewer sinks. Since the sinks are better positioned, it also needs fewer relays. The runtime of GRASP-MSRP increases from 196.5039 seconds for 25 candidate sinks to 826.5791 seconds for 81 candidates.

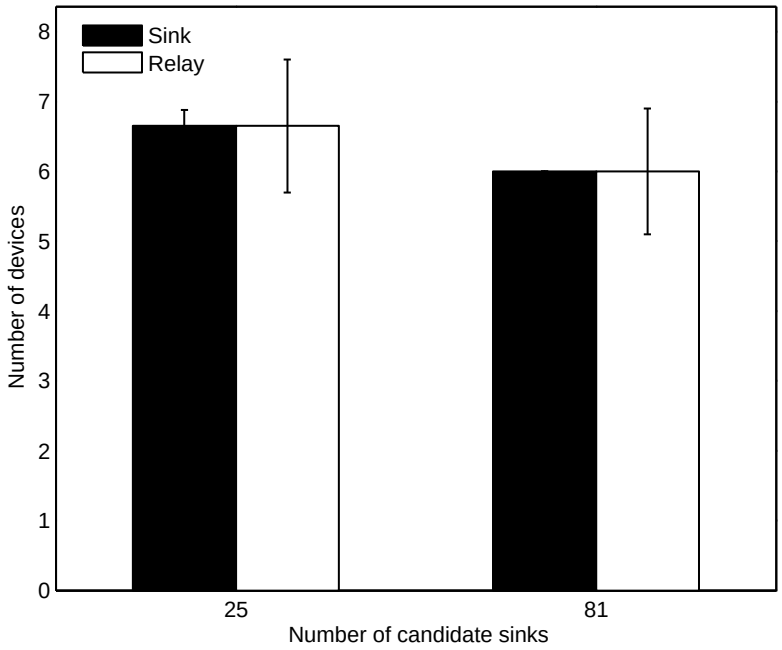


Figure 94: Total numbers of sinks and relays for GRASP-MSRP versus number of candidate sinks

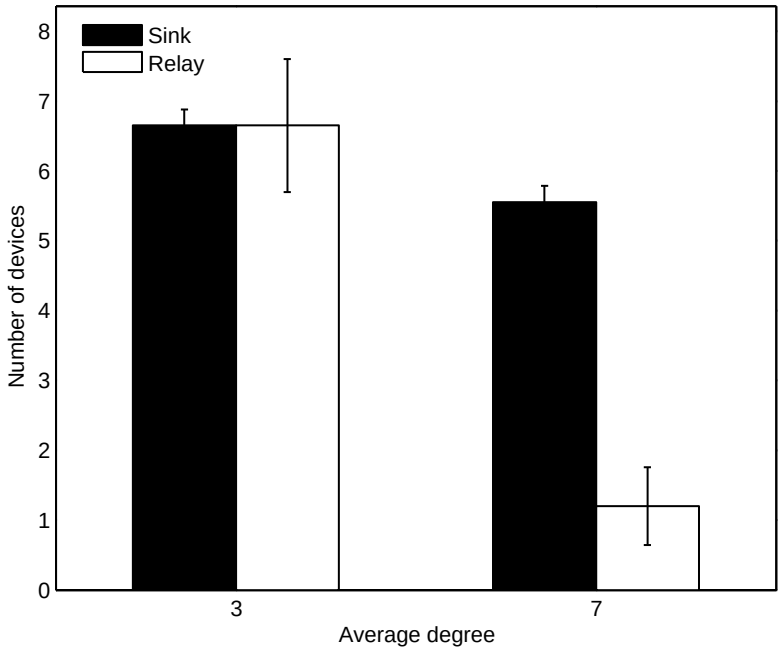


Figure 95: Total numbers of sinks and relays for GRASP-MSRP versus average degree

Finally, we evaluate the performance of GRASP-MSRP when the number of sensor nodes increases. In the first case, we increase the number of sensor nodes from 100 to 200 while keeping the area fixed. This affects the density of the network, where the average degree of a sensor node increases from 3 to 7. For this simulation, we use 25 candidate sinks and 81 candidate relays. When the network becomes denser, the numbers of required sinks and relays decrease as depicted in Figure 95. This happens because when the network density increases, path lengths from a sensor node to sinks become shorter and a sensor node has more neighbours that help finding alternate routes to sinks when it fails. However, GRASP-MSRP takes longer time to compute the solution, i.e. from 196.5039 seconds to 319.2039 seconds when the average degree increases.

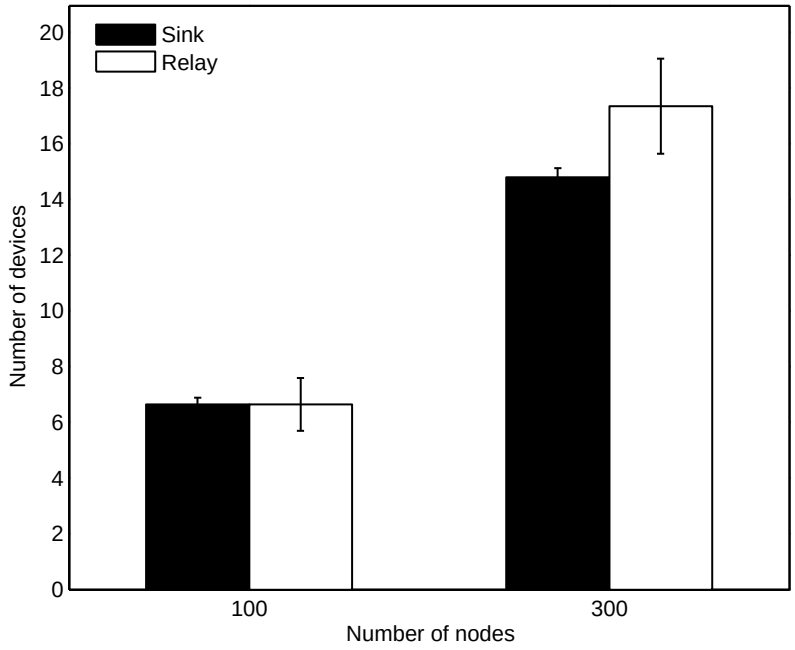


Figure 96: Total numbers of sinks and relays for GRASP-MSRP versus number of nodes

In the second case, we increase the number of sensor nodes from 100 to 300 while keeping the average degree fixed, so we enlarge the area. As a result, we need more candidate sinks and relays. In this simulation, we use 81 candidate sinks and 225 candidate relays for 300 sensor nodes. When the network area becomes bigger, more

sinks and relays are required to be deployed as shown in Figure 96. The runtime of GRASP-MSRP increases from 196.5039 seconds to 19,648.8984 seconds in this simulation.

7.6 Conclusion

We define the problem of increasing network robustness by protecting it against one single failure, of either a sink or a sensor node. We design a network to be double-covered and non-critical. Double-covered means each sensor node must have at least two length-bounded paths to two sinks. Non-critical means all sensor nodes must have length-bounded paths to sinks when an arbitrary sensor node fails. Our novel contributions are solutions to minimise the deployment cost of sinks and relays.

We firstly look at the multiple sink placement problem and propose Greedy-MSP and GRASP-MSP to minimise the total sink cost. Both algorithms solve the multiple sink placement problem by ensuring that each sensor node in the network is double-covered. Empirically, Greedy-MSP has the shortest runtime, but GRASP-MSP achieves comparable cost to the optimal solution with shorter runtime. GRASP-MSP's simulation results justify the use of local search to solve the multiple sink and relay placement problem, where a linear optimal solution is not available.

We then propose Greedy-MSRP and GRASP-MSRP to solve the multiple sink and relay placement problem, where we want the designed topologies to be double-covered and non-critical. Our simulation results show that the k-means clustering-based algorithms outperform Greedy-MSRP in terms of lower cost solution and shorter runtime because they have better sink positions. On the other hand, GRASP-MSRP outperforms the other algorithms with the lowest cost solutions and the shortest runtime. The GRASP-MSRP results also show that more sinks are exchanged with relays when the sink cost increases to reduce the total deployment cost.

In this chapter, if both the number of uncovered nodes and the number of critical nodes in a network are zero, we guarantee robustness against single failure. However, we expect to have benefit for multiple failures, which will be discussed in Chapter 8.

Chapter 8

Evaluation of Network Performance

8.1 Introduction

In this chapter, we evaluate the network performance for each designed Wireless Sensor Network (WSN) topology. Firstly, we want to show that networks with additional relays are more robust to failures than the original topologies. We also want to compare the performance of topologies that have more relays to topologies that have fewer relays. Secondly, we want to evaluate the robustness and scalability of topologies that have multiple sinks, which are expected to tolerate more node failures.

We have presented algorithms to deploy additional relays, namely GRASP-ARP in Chapter 5 and GRASP-ABP in Chapter 6, as well as to deploy sinks and relays, that is GRASP-MSRP in Chapter 7. The objective of these algorithms is to minimise the deployment cost of a WSN with faster runtime. However, having minimal cost with faster runtime does not necessarily mean we have a robust solution. In this chapter, we simulate the resulting networks from those algorithms in ns-2 [2], killing some nodes, and measuring the performance of the designed topologies under

network operations. Ns-2 is a discrete-event network simulator widely used for WSN and other network simulations. We simulate multi-hop data gathering using our proposed ER-MAC from Chapter 4 with its forward-to-parent routing mechanism.

8.2 Preliminary Discussions and Details of Simulation

In this chapter, we use the following metrics to evaluate the network performance in ns-2 simulations:

1. ***Packet delivery ratio*** measures the number of packets successfully received by the sink over the number of packets generated by source nodes. Since the networks with relays are expected to have higher connectivity than the original topologies when some nodes fail, the sink is also expected to receive more packets from the source nodes.

2. ***Average per packet latency*** measures the average per packet transmission time through a multi-hop network. The latency for networks with relays should be shorter than the original topologies because relays help sensor nodes forward traffic and may shorten the routing paths for some sensor nodes. When some nodes fails, relays provide alternate routes to the sink. However, due to the time needed to find new routes, packets are buffered and the latency is expected to increase.

3. ***Connectivity*** measures the percentage of alive sensor nodes that are still connected to the sink through multi-hop communication. As sensor nodes or relay nodes fail, we expect that the networks with relays will have higher connectivity when compared with the original networks.

8.2.1 Preliminary Discussions

We evaluate networks with one sink in Section 8.3. For the single sink scenario, we compare the performance of the original topologies to the topologies with additional relay nodes, where we use the resulting topologies from GRASP-ARP and GRASP-ABP. When several nodes fail, either sensor nodes or relay nodes, the GRASP-ARP and GRASP-ABP topologies with additional relays show improvements over the original topologies. Specifically, the GRASP-ARP topologies achieve the best results as they have more relays that guarantee 2-connectivity to the sink compared to GRASP-ABP. With the GRASP-ARP topologies, the number of sensor nodes that are still connected to the networks after some failures increases by more than 30%, while the number of packets delivered increases by up to 35% compared to the original topologies. With the GRASP-ABP topologies, the improvements are around 25% for connectivity and 30% for delivery ratio. Furthermore, to show that the improvements of the network performance is not specific to ER-MAC, we also evaluate the network performance using Z-MAC [97]. For Z-MAC, we use Shortest Path Tree Routing as the routing protocol. In the simulation, Z-MAC also shows improvements for topologies with relays compared to the original topologies. It achieves up to 20% higher connectivity and 10% higher delivery ratio.

The performance evaluation of networks with multiple sinks is discussed in Section 8.4. In the first set of simulations, we evaluate networks with four sinks deployed at the four corners of the networks. We compare the original topologies to the topologies with additional relays that resulted from GRASP-ARP and GRASP-ABP. When the networks have multiple sinks, the topologies of GRASP-ARP and GRASP-ABP show comparable performance and outperform the original topologies by around 20% for both connectivity and delivery ratio. In the second simulation set, we compare the GRASP-ABP topologies, given that they have similar results to GRASP-ARP in the four corner sink scenario, to the topologies that resulted from GRASP-MSRP with variable numbers of sinks. The topologies of both GRASP-ABP and GRASP-MSRP have additional relays. While the GRASP-ABP

topologies have four sinks fixed at the four corners of the networks, GRASP-MSRP deploys sinks at the best locations. In the simulation, the GRASP-MSRP topologies with three sinks achieve around 5% improvement in connectivity and delivery ratio compared to the GRASP-ABP topologies with four sinks after several failures. This confirms the importance of not only having multiple sinks in the networks, but also placing them at the best positions.

8.2.2 Details of Simulation

We take the resulting topologies generated in the previous chapters and deploy, in simulation, sensor nodes, relay nodes and sinks according to the deployment plans. The topologies are evaluated in ns-2 using the proposed ER-MAC from Chapter 4 with its forward-to-parent routing mechanism. Because our purpose in this simulation is to evaluate the designed topologies, not the communication protocol, we assume the no-fire situation only. The simulation results that compared the no-fire and in-fire situations were presented in Chapter 4. Recall that in the no-fire situation, nodes can only send packets in their own transmit slots.

The simulation results presented are based on the average of five topologies that are simulated five times each. Note that we do not show error bars in line graphs to improve their readability. In each experiment, we simulate a data gathering, where all sensor nodes are the source nodes that generate packets with a fixed interval. They also forward other nodes' packets toward the sink. As ER-MAC supports packet prioritisation, we force source nodes to generate one high priority packet and one low priority packet at the same time every 20 seconds. Therefore, the traffic load is 0.1 packets/node/sec. Relay nodes do not generate packets, but only forward them, and are used from the start of the simulation. Our works assume point-based failures, where the failed devices are scattered in the network. We do not assume that relay nodes are more robust than sensor nodes, so they too may fail during the simulation period. In the simulation, we increase the number of dead nodes gradually by killing one node, either a sensor or a relay, in each time step.

Since a common node problem is node death due to energy depletion, which is either caused by normal battery discharge, short circuits or leakage due to broken packaging [18], we consider the probability of node death in our simulation to be proportional to the work done. That is, instead of selecting dead nodes randomly, we bias the node death towards the nodes that have done most work. Firstly, we list all alive nodes (sensors and relays) in increasing order of energy consumption, count the total energy used, and calculate a weight for each node by using the ratio of own-energy to the total-energy. We then use these weights to generate an array of numbers between 0 and 1 representing bins of different size. The weight of a bin that associates with node a is the total weight of all nodes whose weight $\leq a$'s weight, including a. To select a node to be killed, we generate a random number between 0 and 1, and select the first bin whose weight is larger than the generated number. Doing it this way means, for example, if node a has 3 times the energy consumption of node b, the bin size for node a is always 3 times the bin size for node b, and so is always 3 times as likely to be killed. That allows us to justify our approach by saying the probability of node death is proportional to the work done.

With this model, it is likely that nodes closer to the sink are the first to die because they have much higher relay loads and drain the batteries quickly. In a low density WSN, the network may get disconnected from the sink if the number of dead nodes increases. However, it is expected that a network partition is a lot less likely to occur in a topology with relays compared to the original one.

8.3 Evaluation of Network Topologies with Multiple Sources and One Sink

In this section, we compare the original topologies, i.e. topologies with no relays, to the topologies with relays generated by GRASP-ARP and GRASP-ABP. Firstly, we want to show that networks with relays are more robust to failures than the

networks without relays. Secondly, we want to show that the topologies generated by GRASP-ABP have comparable performance to the topologies that resulted from GRASP-ARP, which have more relay nodes. We use the original topologies, which are 100-node networks (average degree 3.2) and their resulting topologies generated by GRASP-ARP with the dynamic programming variant of Counting-Paths for 2-connectivity, GRASP-ABP with Connectivity Threshold (CT) and Rerouting Threshold (RT) equal to 0%, and GRASP-ABP with both thresholds equal to 2%. Therefore, the topologies with relays can only guarantee robustness against one failure and are expected to lose performance when the number of failures increases. In all topologies, the sink is located at the top-left corner of the networks. In each experiment, we simulate a data gathering for 6,000 seconds and kill one node every 1,000 seconds, either a sensor node or a relay node, start from the $1,000^{\text{th}}$ second. We gather statistical data every 1,000 seconds and plot the results. Therefore, the statistics after one node dies are plotted at the $2,000^{\text{th}}$ second.

8.3.1 Experiments Using ER-MAC

Figure 97 shows the linear drop of high priority packets' delivery ratio in all topologies when the number of failed nodes increases. Even if the schedules have been readapted between node killings, a network with ER-MAC continues to lose packets because of three reasons: the dead node is not recognised instantly, packets are dropped when nodes wait to readapt new schedules, and no alternate routes found. In ER-MAC, data packets are transmitted in contention-free slots, so they are not acknowledged. When a node dies, its direct children keep sending packets to it in their transmit slots. They are not aware that the parent node is dead until several data gathering cycles because they do not receive any synchronisation messages from it. The dead node's direct children also have to buffer packets from their descendants while waiting to readapt new schedules. They drop packets when their queues are overload. Packets are also counted as lost if sensor nodes keep generating packets but no alternate routes are found after their parent dies.

Figure 97 also shows the delivery ratio improvement for networks with relays over the original networks. Bear in mind that the topologies are designed to tolerate one failure only. The GRASP-ARP topologies achieve the highest performance because the networks not only have more relays but also guarantee 2-connectivity to the sink. The results shown at the $2,000^{\text{th}}$ second are the statistics after one node dies and GRASP-ARP already achieves around 15% improvement over the original topologies. The largest gap is achieved when four nodes die at the $5,000^{\text{th}}$ second, which is up to 35%. GRASP-ABP with 0% threshold deploys fewer relays than GRASP-ARP, so its delivery ratio is slightly lower than GRASP-ARP's, but the gap is not more than 10%. Even though GRASP-ABP with 2% threshold deploys the fewest relays and offers the lowest delivery ratio compared to GRASP-ARP and GRASP-ABP with 0% threshold, it still outperforms the original networks by around 7% to 17%. If no additional nodes die, the delivery ratios are not expected to improve much because if a network is partitioned, the disconnected part still generates packets that will never be delivered to the sink. If nodes continue to fail, the observed trends continue, where the networks with relays have better delivery ratio compared to the original topologies. However, when the sink is disconnected, the delivery ratios saturate.

The simulation results for the low priority packets' delivery ratio are presented in Figure 98. ER-MAC is designed to prioritise high priority packets. Therefore, the overall delivery ratio of low priority packets is not as high as the high priority ones as expected, but it still shows an improvement for networks with relays. The figure shows comparable results for the topologies of GRASP-ARP and GRASP-ABP with 0% threshold, followed by GRASP-ABP with 2% threshold and the original topologies.

The average per packet latency for high and low priority packets are depicted in Figure 99 and Figure 100, respectively. Firstly, we will explain why the initial latency of high priority packets in Figure 99 is around 20 seconds when there are no failed nodes. Recall that we simulate the normal mode of ER-MAC for the no-fire

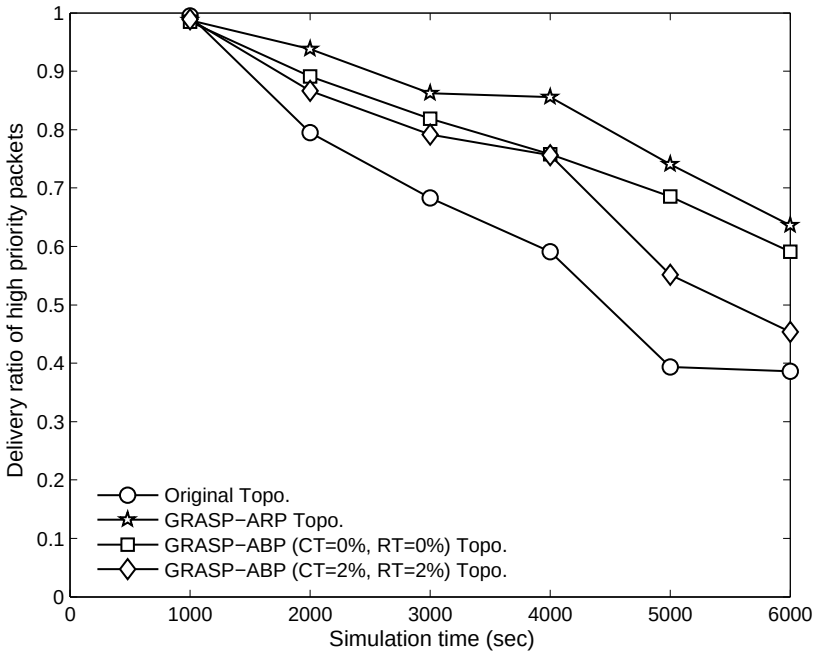


Figure 97: Delivery ratio of high priority packets for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds

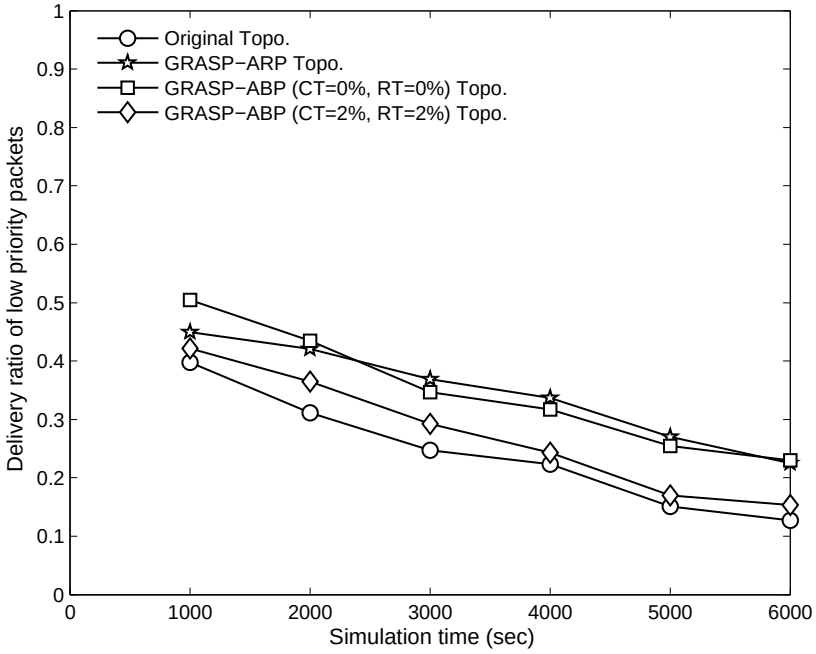


Figure 98: Delivery ratio of low priority packets for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds

situation, where the communication is delay-tolerant. In the normal mode, nodes sleep most of the time and each node can only send one of its packets in one data gathering cycle. Since the network has one sink, there is only one routing tree and the duration of one data gathering cycle in a 100-node network in this simulation is more than 15 seconds. This duration dominates the latency and is equivalent to the number of collision-free slots in one TDMA frame times the slot size.

The simulation results also show that the latency of low priority packets in Figure 100 is higher than the latency of high priority packets in Figure 99. This happens due to the queuing delay of the low priority packets when ER-MAC prioritises the high priority ones. The latency increases when nodes die because ER-MAC buffers packets when the routing tree is reconfigured and the TDMA schedules are rebuilt. The latency drop in the original topologies corresponds to the low delivery ratio, because only nodes closer to the sink can deliver their packets when the networks become disconnected.

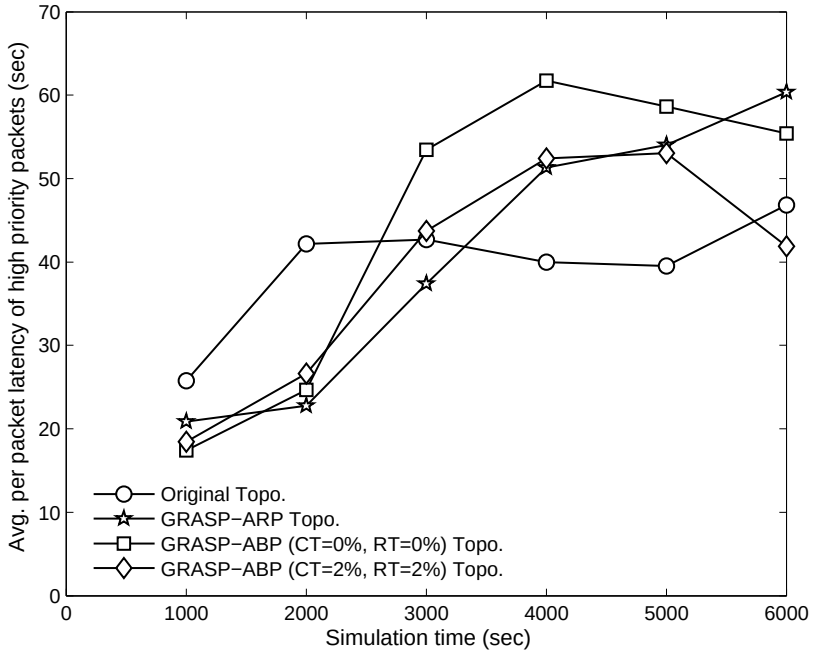


Figure 99: Latency of high priority packets for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds

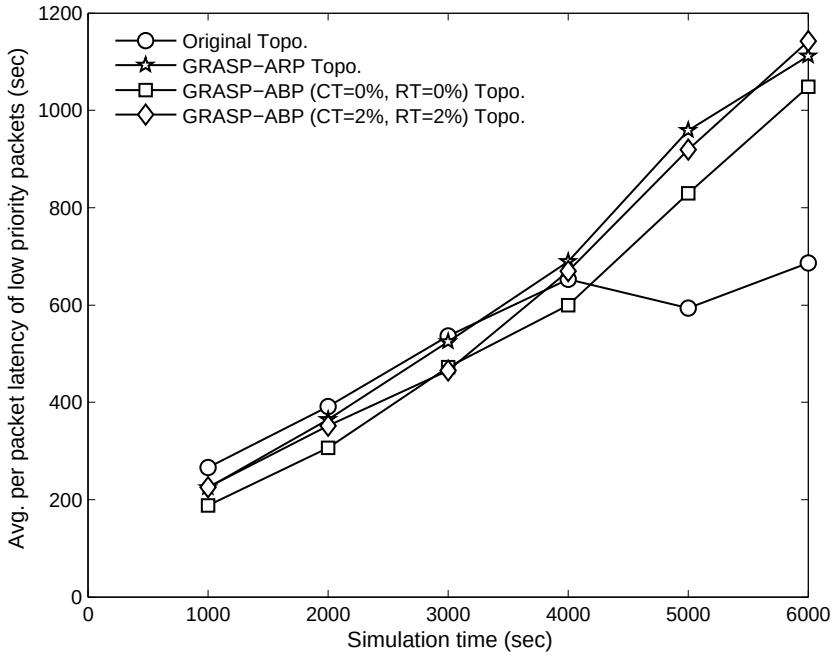


Figure 100: Latency of low priority packets for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds

Connectivity calculates the percentage of alive sensor nodes that are still connected to the sink. A sensor node is counted as connected if at least one of its generated packets is received by the sink. As depicted in Figure 101, the connectivity performance follows similar trends as the delivery ratio in Figure 97, because a sensor node is counted as connected if one of its packets is received by the sink. The GRASP-ARP topologies achieve around 32% improvement in connectivity over the original topologies after five nodes fail, followed by GRASP-ABP with 27% improvement for the 0% threshold and 10% improvement for the 2% threshold. At this stage, we have shown the trade-off between the number of additional relay nodes and the network performance for the single sink scenario using our proposed ER-MAC protocol. We show that the networks with relays have better performance than the original networks. When a network has more relays, it is more robust to failures. The GRASP-ARP topologies have the best performance because they have more relays that guarantee 2-connectivity to the sink compared to the GRASP-ABP topologies,

but GRASP-ABP generates topologies faster than GRASP-ARP and requires fewer relays.

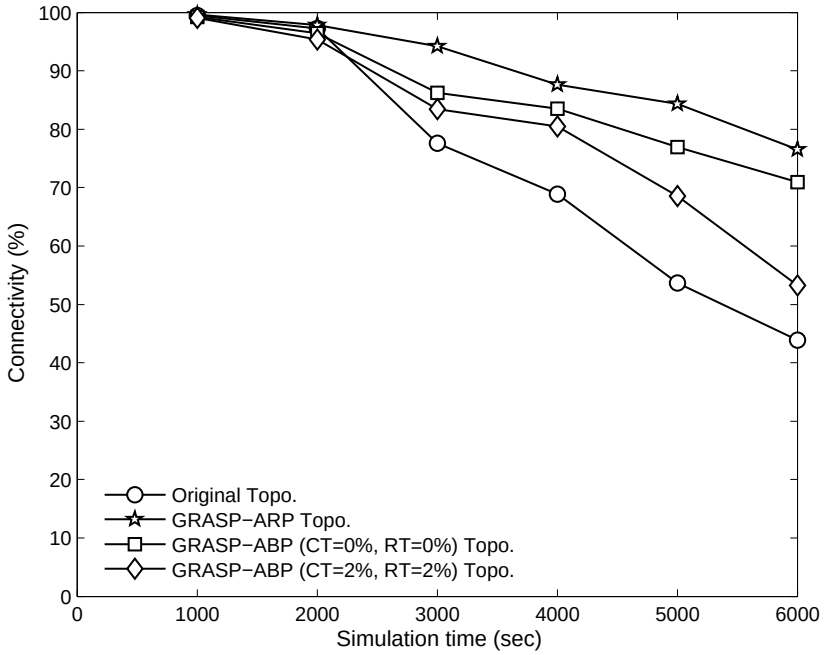


Figure 101: Connectivity for multiple sources – single sink with ER-MAC where a node dies every 1,000 seconds

8.3.2 Experiments Using Z-MAC

To show that the network performance is not specific to ER-MAC, we also simulate the data gathering application using Z-MAC [97]. In this simulation, Z-MAC can adaptively switch between Low Contention Level (LCL) and High Contention Level (HCL) based on packet loses due to hidden terminals. In LCL mode, nodes can contend in any time slots. However, in HCL mode, only the owner of the slot and one-hop neighbours of the owner of the slot can contend for the slot. Since there is no packet prioritisation in the Z-MAC design, we do not distinguish between high and low priority packets. For each sensor node, we generate two packets every 20 seconds to keep the same traffic load as the simulation with ER-MAC, which is 0.1 packets/node/sec.

For simulations with Z-MAC, we use Shortest Path Tree Routing (STR) as the routing protocol. STR is similar to the routing protocol from Collection Tree Protocol (CTP) [47], which has two routing mechanisms, i.e. data path validation and adaptive beaconing, with neighbour discovery ability after a node's parent in the routing tree dies. The two mechanisms enable the routing protocol to be robust to stale route information and agile to link dynamics. However, CTP uses expected transmissions as the cost metric, while our implementation of STR uses hop counts. STR forwards packets using the shortest route toward the sink. The routing decisions are made based on local information, where a node selects a parent from its one-hop neighbours that has the smallest hop count to the sink.

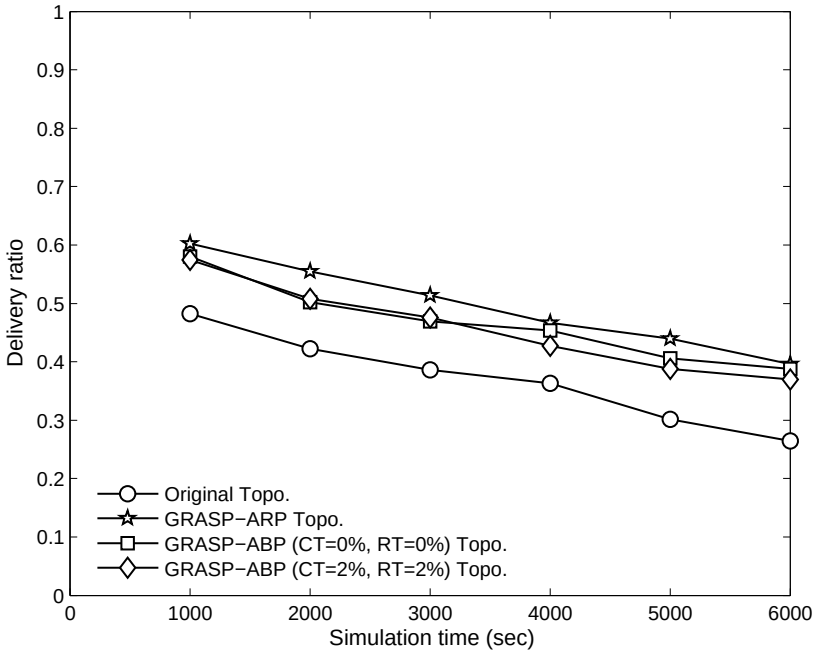


Figure 102: Delivery ratio for multiple sources – single sink with Z-MAC where a node dies every 1,000 seconds

Figure 102 presents the delivery ratio of Z-MAC in various topologies. As has been shown in Chapter 4, Z-MAC's delivery ratio is lower than ER-MAC's. However, Z-MAC also shows improvements in the topologies with relays, i.e. around 10% higher delivery ratio than the original topologies. Network connectivity with Z-MAC is

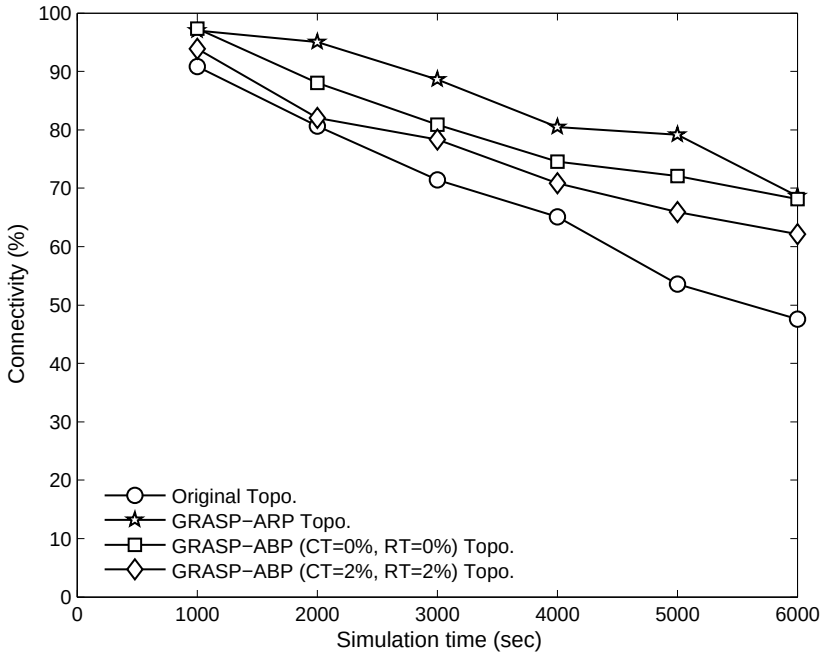


Figure 103: Connectivity for multiple sources – single sink with Z-MAC where a node dies every 1,000 seconds

shown in Figure 103. With the topologies of GRASP-ARP and GRASP-ABP with 0% threshold, Z-MAC improves the connectivity of the networks by 20% after five nodes fail. At this stage, we have shown that the improvements in the network performance for topologies with relays is not specific only to ER-MAC. Because the performance of the networks with Z-MAC follows similar trends as with ER-MAC, where topologies with more relays that guarantee 2-connectivity to the sink have better results, we will only use ER-MAC in our further simulations.

8.4 Evaluation of Network Topologies with Multiple Sources and Multiple Sinks

In this section, we evaluate the performance of the networks with multiple sinks. We firstly look at the scenario where four sinks are placed at the four corners of the networks. We choose to put the sinks at the corners, which have lower connectivity,

so the topologies are easier to partition. Then in the second scenario, we compare the performance of the networks with variable numbers of sinks. We use ER-MAC in this experiment and follow the same simulation setup as in the previous section. However, in this experiment, we simulate data gathering for 3,000 seconds only, because the period of one data gathering cycle in networks with many sinks is shorter than in the single sink problem. We increase the number of dead nodes by killing one node every 250 seconds, either a sensor node or a relay node.

8.4.1 Evaluation of Network Topologies with Four Sinks

In the simulations with four sinks, we compare the original topologies, which are 100-node networks (average degree 3.2) and their resulting topologies generated by GRASP-ARP with the dynamic programming variant of Counting-Paths for the any-sinks cases, GRASP-ABP with Connectivity Threshold (CT) and Rerouting Threshold (RT) equal to 0%, and GRASP-ABP with both thresholds equal to 2%. In all simulated topologies, we fix the locations to place the four sinks at the top-left, top-right, bottom-left, and bottom-right corners of the networks.

Figure 104 shows the delivery ratio of high priority packets and Figure 105 shows the delivery ratio of low priority ones. Firstly, we see improvements over the delivery ratios of both high and low priority packets in the single sink scenario, which are presented in Figure 97 and Figure 98, respectively. This proves the advantages of having multiple sinks, i.e. the networks become more scalable and more robust compared to the traditional one sink networks. With multiple sinks, the delivery ratios of both high and low priority packet in all topologies after five nodes die are above 0.7. Secondly, while the topologies of GRASP-ARP always achieve the best results in the single sink scenario, having more than one sink deployed makes the topologies of GRASP-ABP with 0% threshold perform similarly to GRASP-ARP's with more relays. They outperform the delivery ratio of the original topologies by around 20% for both high and low priority packets after 11 nodes die.

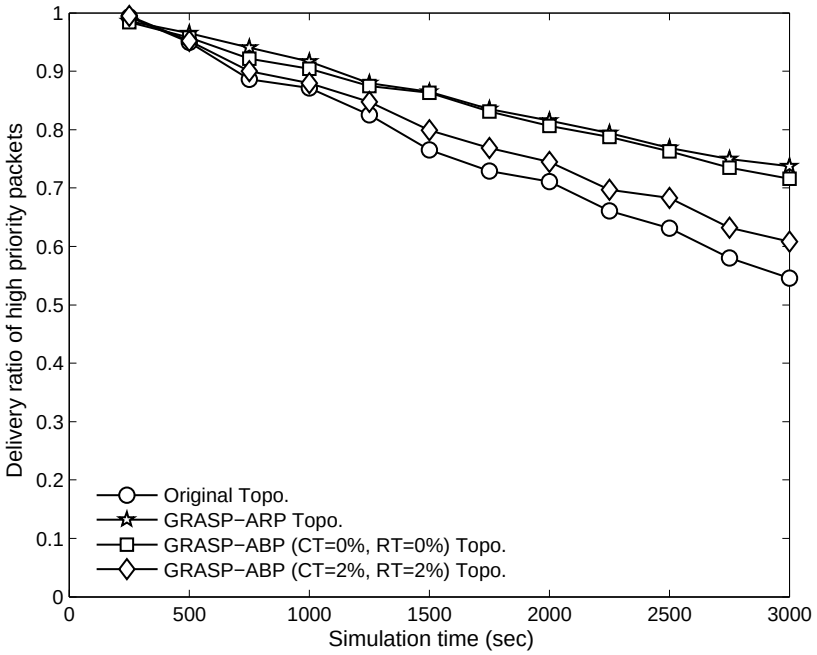


Figure 104: Delivery ratio of high priority packets for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds

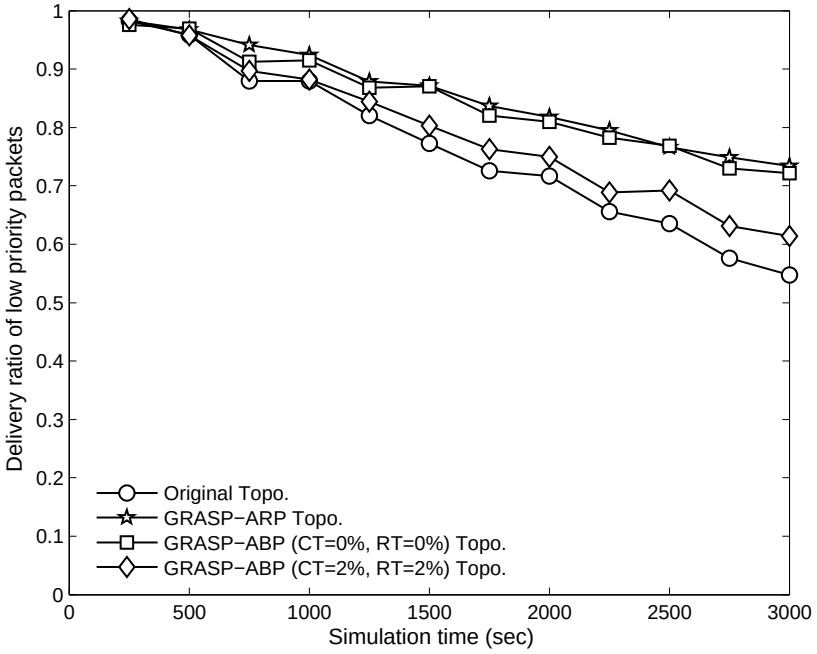


Figure 105: Delivery ratio of low priority packets for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds

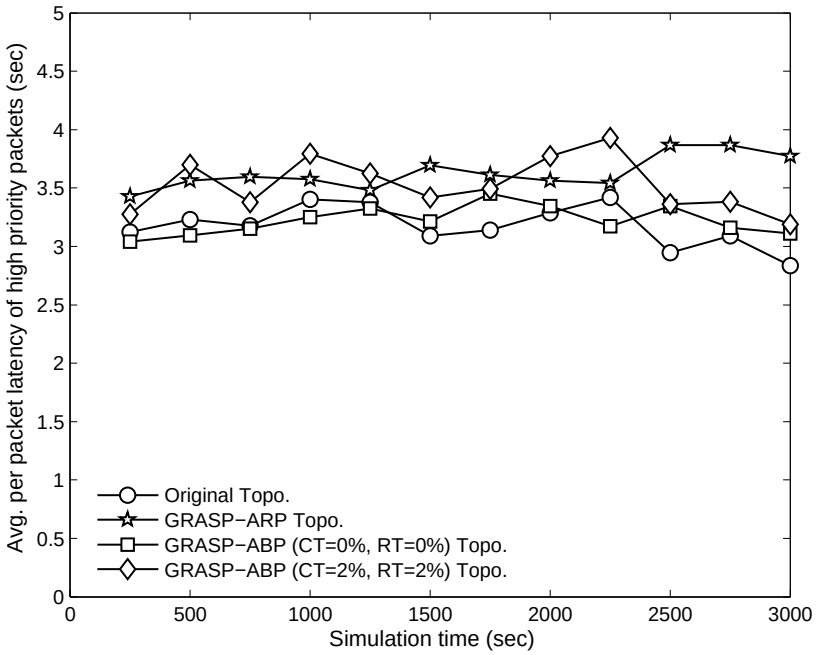


Figure 106: Latency of high priority packets for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds

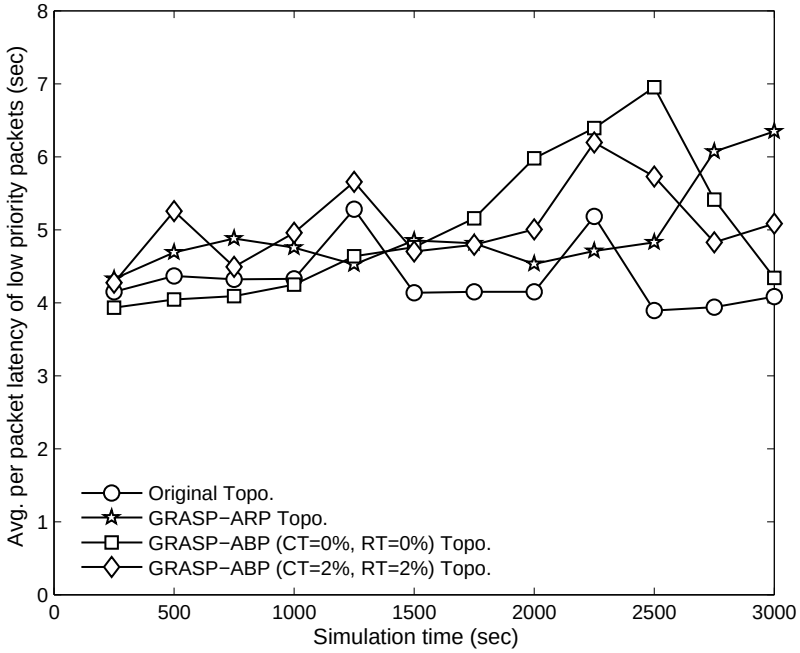


Figure 107: Latency of low priority packets for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds

The latency of high and low priority packets are presented in Figure 106 and Figure 107, respectively. Having multiple sinks makes the networks more scalable, because of shorter hop counts to reach the nearest sinks and the availability of more alternate routes when nodes die. This results in a significant drop in latency for the two kinds of packets. The latency of high priority packets in all topologies is in a range between 2.8 and 4 seconds, while it is between 3.8 and 7 seconds for the low priority ones. Moreover, the latencies do not increase significantly when nodes fail, because the routing trees in the multiple sink scenario are smaller, so the tree reconfiguration when a node dies is faster than in the single sink scenario.

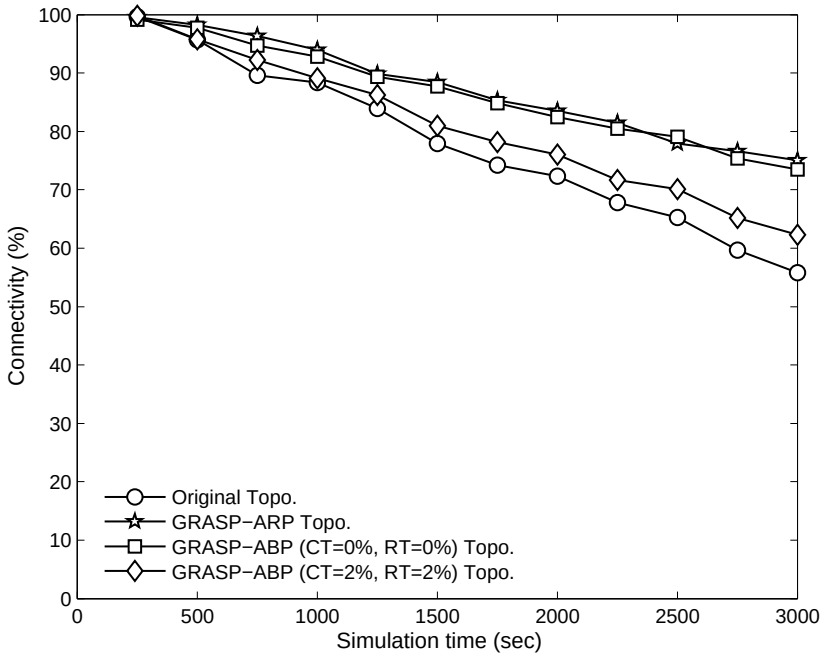


Figure 108: Connectivity for multiple sources – four sinks with ER-MAC where a node dies every 250 seconds

When the networks have multiple sinks, the topologies of GRASP-ARP and GRASP-ABP with 0% threshold have similar connectivity as shown in Figure 108, that is 20% higher than the original topology's connectivity. This result corresponds to the delivery ratios as shown in Figure 104 and 105. With the fewest deployed relays, the topologies of GRASP-ABP with 2% threshold also improve on the connectivity of

the original topologies by almost 10% after the failure of 11 nodes. From this simulation set, we can infer that having many deployed sinks increases the robustness and scalability of the networks. We show this in the experiment by higher delivery ratios, latency and connectivity compared to the single sink scenario. Moreover, we also show that the topologies of GRASP-ARP and GRASP-ABP with 0% threshold have similar network performance in the simulation.

8.4.2 Evaluation of Network Topologies with Variable Numbers of Sinks

Given that the GRASP-ABP with 0% threshold topologies have similar results to the GRASP-ARP's, in the second simulation set, we only compare the GRASP-ABP topologies to the topologies that resulted from GRASP-MSRP with variable numbers of sinks. All topologies of GRASP-ABP and GRASP-MSRP have relays, but GRASP-ABP has four fixed sinks at the four corners of the networks, while GRASP-MSRP places sinks at the best candidate locations. In the experiment, we use topologies of GRASP-MSRP with three and six sinks. With three sinks, the maximum path length to the nearest sink for every sensor node ($l_{\max}$) is 10 hops. For six sinks, it is six hops.

We will present the results for the high priority packets only because the low priority packets' results generally follow similar trends as have been shown in the previous simulation set. The delivery ratio of high priority packets while nodes are failing is depicted in Figure 109, where the GRASP-MSRP topologies with six sinks achieve the highest ratio. The second highest delivery ratio is not achieved by the topologies of GRASP-ABP with four sinks, but by GRASP-MSRP with three sinks. The delivery ratio gap between these two simulation results is around 5% after several failures. From this experiment, we not only show that having more sinks gives us better performance, but also that placing them at the best locations is more important.

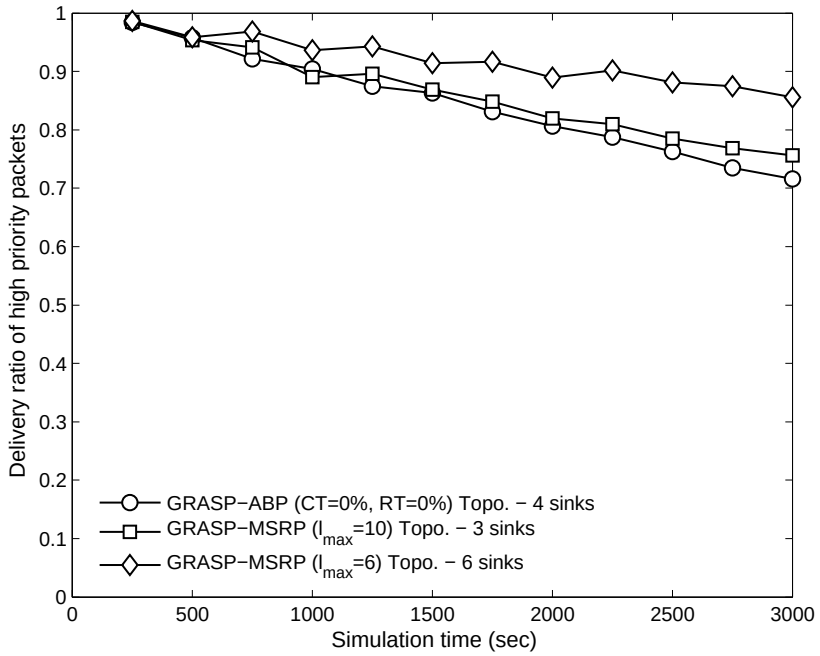


Figure 109: Delivery ratio of high priority packets for multiple sources – variable numbers of sinks with ER-MAC where a node dies every 250 seconds

The latency of high priority packets is shown in Figure 110. As expected, the topologies of GRASP-MSRP with six sinks have the lowest latency, i.e. around 0.6 second, followed by the topologies with three sinks, i.e. 2 seconds in average. The latency of the GRASP-ABP topologies with four sinks is slightly higher than 3 seconds because most of the nodes have longer paths when the sinks are deployed at the corners of the networks.

The network connectivity for this simulation set is presented in Figure 111. This results correspond with the delivery ratio as shown in Figure 109, where the topologies of GRASP-MSRP with six sinks offer the best performance, followed by the topologies with three sinks. The GRASP-ABP topologies with four sinks have 5% lower connectivity than the topologies of GRASP-MSRP with three sinks after several failures due to the sinks' positions. In this experiment, we show that if we have higher budget to deploy more sinks, we can get better network performance, especially if we place them at the best locations.

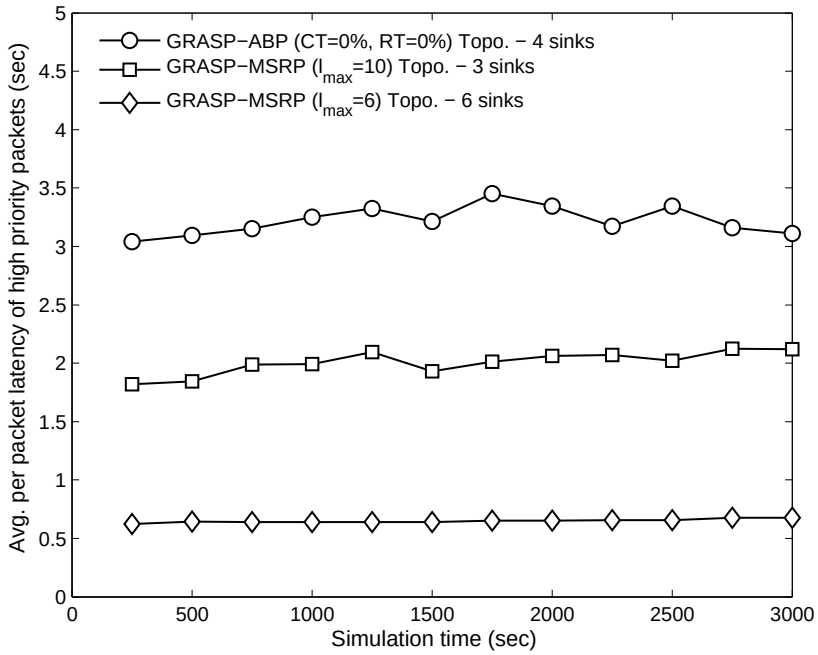


Figure 110: Latency of high priority packets for multiple sources – variable numbers of sinks with ER-MAC where a node dies every 250 seconds

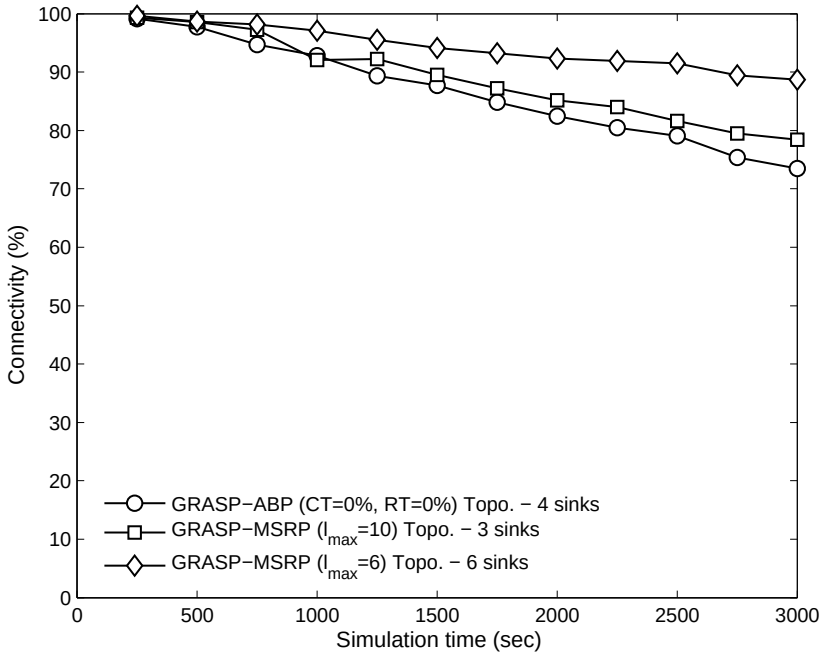


Figure 111: Connectivity for multiple sources – variable numbers of sinks with ER-MAC where a node dies every 250 seconds

8.5 Conclusion

In this chapter, we take the original topologies and their resulting topologies generated by GRASP-ARP, GRASP-ABP and GRASP-MSRP, and simulate data gathering applications using ER-MAC and Z-MAC in ns-2 to measure the network performance. By this experiment, we want to show the trade-off between having low cost networks, in terms of the numbers of relays and sinks, and having robust solutions when some nodes fail. We simulate several scenarios: one sink, four corner sinks and variable numbers of sinks. From the simulations with one sink and four corner sinks, the results show that networks with more relays, i.e. the GRASP-ARP topologies that guarantee 2-connectivity, achieve better performance. Even though the topologies of GRASP-ABP with 0% threshold retreat from the network-wide 2-connectivity and have fewer relays than GRASP-ARP, their performance are comparable. Moreover, the topologies generated by GRASP-ABP with 2% threshold also have better performance than the original topologies.

Having multiple sinks makes the networks more robust to failures and scalable, i.e. have higher delivery ratio, lower latency and higher connectivity, compared to the single sink networks. In the multiple sink scenario, the performance of a network is not only influenced by the number of deployed sinks, but more importantly the positions to deploy the sinks. As we can see from the simulations with variable numbers of sinks, the best performance is achieved by topologies with six sinks where the placements were optimised (GRASP-MSRP). Note that topologies with only three sinks but where the placements were optimised (GRASP-MSRP) are better than topologies with four sinks in fixed positions (GRASP-ABP).

Chapter 9

Conclusion and Future Work

9.1 Summary of Contributions

In this thesis, we demonstrate that Medium Access Control (MAC) protocols and topology planning algorithms can be designed together to create fault-tolerant Wireless Sensor Networks (WSNs) that trade-off robustness and deployment cost. Fault-tolerance is important for many WSN applications, where messages from all sensor nodes must be delivered to base stations or sinks in a reliable and timely manner. To ensure reliable delivery, a WSN must be able to cope gracefully with failures that are caused by dropped packets due to overload, node/link failures, and disconnected networks, for example during emergency response. In WSNs, successful packet transmission is determined by a MAC protocol. To be robust to failures, firstly the MAC protocol must be able to adapt to traffic and topology changes. Secondly, the physical network topology must ensure that alternate routes to the sinks are in fact available. This requires topology planning that guarantees the existence of alternative routes from all sensor nodes to the sinks, which can achieved by finding the best locations to deploy sinks and some additional relays. We summarise the main contributions of our solutions and achievements below.

9.1.1 A Hybrid MAC Protocol for Emergency Response

In Chapter 4, we focus on the MAC protocol design for emergency response WSNs. As an emergency situation rarely occurs, a WSN spends most of its lifetime in normal monitoring. During normal monitoring, the traffic load is light and the communication is delay-tolerant, so the main objective of the protocol is energy efficiency. However, the protocol must be able to switch from normal to emergency monitoring when a hazard event is detected. During emergencies, rapid reliable receipt of critical data is most important, and so energy efficiency should be traded for high delivery ratios and low latency. We present ER-MAC, a MAC protocol for emergency response WSNs. ER-MAC is novel as it tackles all of the following design criteria: operates efficiently during normal monitoring, achieves high delivery ratio and low latency for emergency response, adapts to traffic and topology changes, prioritises high priority packets and supports fairness over the packets' sources.

ER-MAC is designed as a hybrid of the TDMA and CSMA approaches, giving it flexibility to adapt to traffic and topology changes. ER-MAC's TDMA characteristic enables it to establish collision-free slots, which are used during normal monitoring to schedule packet transmissions. Furthermore, its CSMA characteristic allows nodes that participate in emergency monitoring to contend in each slot to achieve high delivery ratio and low latency. ER-MAC is also designed with two queues to buffer high and low priority packets separately. In its operation, ER-MAC prioritises the high priority packets and sacrifices the delivery ratio and latency of the low priority ones. In addition, its synchronised and loose slot structure enables nodes to modify their schedule and perform local repair. This ability helps nodes join or leave the network.

We implement and simulate ER-MAC in ns-2. The results show that ER-MAC outperforms Z-MAC, another hybrid MAC protocol that also allows contention in TDMA slots, by higher delivery ratio and lower latency at low energy consumption. Moreover, the performance of ER-MAC is influenced by network topologies,

especially when some nodes fail during the network operation. As other protocols, ER-MAC gives a better performance on topologies that are not easy to partition and have shorter path lengths to the sink.

9.1.2 Fault-Tolerant Relay Deployment for k Vertex-Disjoint Paths

We study fault-tolerant topology planning which only involves additional placement of relays in Chapter 5. To be able to tolerate $k-1$ node failures, each sensor node in the initial design must have k disjoint paths to one or more sinks. In addition, it is also necessary that the lengths of the paths are bounded to minimise packet delay to reach sinks. A WSN is (k, l)-sink-connected if and only if each sensor node has k disjoint paths of length $\leq l$. If a WSN is not (k, l)-sink-connected, additional relays are required.

In this chapter, we propose two offline algorithms to be run during the initial topology planning to solve this problem. Firstly, we propose Counting-Paths, an algorithm that uses the Ford-Fulkerson maximum flow algorithm to count the number of disjoint paths from a sensor node to a sink. To speed up the counting process for an entire network, we implement a dynamic programming variant of Counting-Paths, where we start counting the number of disjoint paths from sensor nodes closer to the sink. The empirical simulation in C++ show that Counting-Paths is more efficient and accurate compared to algorithms from the literature, namely Modified Dijkstra, Fast Pathfinding and Maximum Paths, and the dynamic programming variant scales even better. Although we have not proven that the dynamic programming variant of Counting-Paths ensures the length-bound, all problems in the simulations show that it does obey the length-bound.

Our second algorithm is *Greedy Randomised Adaptive Search Procedure for Additional Relay Placement* (GRASP-ARP). It is a local search algorithm that uses Counting-Paths and modifies the existing GRASP algorithm to deploy the fewest

relays at the possible candidate locations. Our simulations in C++ show that for larger networks GRASP-ARP is significantly faster and requires fewer relays compared to K-CONN-REPAIR, a relay deployment algorithm from the literature. However, requiring (k, l)-sink-connected networks is expensive, because many additional relays are required and simulation runtime increases as we increase the network size. We then try to reduce deployment cost and computation time by focusing only on the important nodes – those whose failure has the worst effect for the network.

9.1.3 Fault-Tolerant Relay Deployment Based on Length-Constrained Connectivity and Rerouting Centrality

In Chapter 6, we continue looking at relay deployment for fault-tolerant WSNs. In order to further reduce deployment cost and computation time, we only deploy relays as backup nodes around important sensor nodes. A sensor node is important to the network if upon its failure, many other sensor nodes will lose their length-bounded paths to the sink.

In this chapter, firstly we propose a solution to identify the important nodes. We define *Length-constrained Connectivity and Rerouting Centrality* (l-CRC) as a new centrality index for WSNs with sinks. l-CRC has a pair of values. The first value measures the importance of a sensor node with respect to network connectivity under a path length constraint, while the second value measures the additional length of shortest paths that would be required after the node fails. Using this centrality index allows us to trade-off deployment cost for robustness. For lower cost deployment, we only address sensor nodes with high centrality to protect the most significant failures. However, by increasing the deployment cost, we can address more sensor nodes with lower centrality and become more robust against more failures.

After identifying sensor nodes which are important, we deploy relay nodes as backups. Instead of deploying relays at the locations nearby the important nodes, we introduce *Greedy Randomised Adaptive Search Procedure for Additional Backup Placement* (GRASP-ABP), a local search algorithm to minimise the number of required relays. We demonstrate empirically in C++ that GRASP-ABP deploys the fewest relays with the shortest runtime compared to GRASP-ARP and K-CONN-REPAIR. In addition, when we raise the centrality threshold, we trade-off the cost of a network against its robustness, and thus decrease the runtime.

9.1.4 Multiple Sink and Relay Placement

While we assume the positions of sinks are given in the previous two chapters, in Chapter 7 we investigate the deployment of both sinks and relays. Still under a path length constraint, we try to protect the network against one single failure, of either a sink or a sensor node, by deploying multiple sinks and relays with minimal cost. For fault-tolerance, we require a WSN to be double-covered and non-critical. Double-covered means each sensor node must have at least two length-bounded paths to two sinks. Non-critical means all sensor nodes must have length-bounded alternative paths to sinks when an arbitrary sensor node fails.

Before investigating multiple sink and relay placement, we first look at the multiple sink placement problem to ensure that each sensor node in the network is double-covered. We propose two algorithms to minimise the total sink cost, namely *Greedy Algorithm for Multiple Sink Placement* (Greedy-MSP) and *Greedy Randomised Adaptive Search Procedure for Multiple Sink Placement* (GRASP-MSP). By using C++, we demonstrate empirically that both algorithms outperform the k-means clustering-based algorithms, namely *Minimise the Number of Sinks for Fault-Tolerance* (MSFT) and *Cluster-Based Sampling for Multiple Sink Placement* (CBS-MSP), and the optimal solution. In simulation, Greedy-MSP has the shortest runtime, but GRASP-MSP achieves the lowest deployment cost. GRASP-MSP's deployment cost is comparable to the optimal solution, but its computation time

is faster. This result justifies the use of local search to solve the multiple sink and relay placement problem, where a linear optimal solution is not available.

We then study the multiple sink and relay placement problem, where we want the network to be double-covered and non-critical. We propose *Greedy Algorithm for Multiple Sink and Relay Placement* (Greedy-MSRP) and *Greedy Randomised Adaptive Search Procedure for Multiple Sink and Relay Placement* (GRASP-MSRP) to minimise the total deployment cost. We also add some modifications to the two k-means clustering-based algorithms and we name them *Minimise the Number of Sinks and Relays for Fault-Tolerance* (MSRFT) and *Cluster-Based Sampling for Multiple Sink and Relay Placement* (CBS-MSRP). These algorithms utilise the concept of Length-constrained Connectivity and Rerouting Centrality (l-CRC) introduced in earlier chapter to identify critical nodes. While Greedy-MSRP, MSRFT and CBS-MSRP deploy sinks using Greedy-MSP, MSFT and CBS-MSP, respectively before placing relays, GRASP-MSRP minimises the number of uncovered and critical nodes simultaneously in its every local search move. The simulation results show that the GRASP-MSRP algorithm has the lowest cost solutions and the shortest runtime. On the other hand, MSRFT and CBS-MSRP outperform Greedy-MSRP by lower cost solution and shorter runtime because they have better sink positions.

9.1.5 Evaluation of Network Performance

After investigating network deployment planning in the previous three chapters, we evaluate the network performance in Chapter 8 to compare the robustness of different topology designs. We take the original topologies and their resulting topologies generated by GRASP-ARP, GRASP-ABP and GRASP-MSRP, and simulate multi-hop data gathering application using ER-MAC and Z-MAC in ns-2. During the simulation, we kill some nodes and measure the performance of the designed topologies. We simulate three scenarios: one sink, four corner sinks and variable numbers of sinks.

For the single sink and four corner sink scenarios, we compare the original topologies, i.e. topologies without relays, to the resulting topologies from GRASP-ARP and GRASP-ABP. When several nodes fail in the single sink scenario, the GRASP-ARP topologies achieve the best performance as they have more deployed relays compared to GRASP-ABP. However, in the four corner sink scenario, the topologies of both GRASP-ABP and GRASP-ARP show comparable results. In addition, having four deployed sinks makes the networks more robust and scalable, i.e. they achieve higher delivery ratio, lower latency and higher connectivity, compared to the single sink networks.

In the variable numbers of sink scenario, we compare the GRASP-ABP topologies, given that they have similar results to GRASP-ARP in the multiple sink scenario, to the topologies that resulted from GRASP-MSRP with variable numbers of sinks. The topologies of both GRASP-ABP and GRASP-MSRP have additional relays. While the GRASP-ABP topologies have four sinks fixed at the four corners of the networks, GRASP-MSRP deploys sinks at the best locations. From the simulations with variable numbers of sinks, the results show that we can have better network performance by not only deploying more sinks in the networks, but also placing them at the best locations.

To sum up, the contributions of this thesis are:

1. ER-MAC, a novel hybrid MAC protocol for emergency response WSNs, that is very energy-efficient in normal monitoring, has high delivery ratio and low latency in emergency monitoring, is traffic and topology adaptive, supports packet prioritisation and gurantees fairness.

2. Counting-Paths, an algorithm to count the number of disjoint paths from a sensor node to a sink, and GRASP-ARP, a GRASP-based local search algorithm that uses Counting-Paths to deploy the fewest relays to guarantee that each sensor node has length-constrained $k \geq 2$ disjoint paths to either one sink

or many sinks. With the dynamic programming variant of Counting-Paths, GRASP-ARP runs faster, but it does not guarantee the length-bound.

3. l-CRC, a new centrality index to identify nodes that need backups, and GRASP-ABP, a local search algorithm that uses l-CRC to minimise the number of required backups (relays) in providing length-bounded alternative paths for the remaining sensor nodes when a sensor fails.

4. Greedy-MSP and GRASP-MSP to deploy multiple sinks with minimal cost to ensure that each sensor node in the network is double-covered, i.e. has at least two length-bounded paths to two sinks.

5. Greedy-MSRP and GRASP-MSRP to deploy multiple sinks and relays with minimal cost to make the network double-covered and non-critical. Non-critical means all sensor nodes must have length-bounded alternative paths to sinks when an arbitrary sensor node fails.

6. Ns-2 simulations using ER-MAC to evaluate the effectiveness of each deployment result, where GRASP-ARP and GRASP-ABP topologies have comparable performance, and the GRASP-MSRP topologies achieve the best results for the multiple sink case because of better sink positions.

9.2 Future Work

In our work on ER-MAC, the design of the protocol is influenced by our assumptions that the WSN has no mobility, all nodes are homogeneous and operate using a fixed transmission range in a single communication channel. We will improve the ER-MAC design by firstly taking into account the ability of sensor nodes to perform multi-channel communications. This will involve new strategies to establish data gathering trees and to build TDMA schedules. We will also consider heterogeneous WSNs, where nodes have different hardware specifications and capabilities. This means sensor nodes may have different transmission ranges and unequal initial

amounts of battery energy. Moreover, we will make improvements on the design to work with both static and mobile nodes, as well as to exploit the ability of nodes to adapt the transmission powers. These two improvements are very important for emergency monitoring as mobile nodes and mobile sinks are expected to join the network. Also, power adaptation enables nodes to increase their transmission ranges to deliver important messages by bypassing routing holes caused by failed nodes or congested areas. In addition, we will include load balancing criteria in our protocol design to reduce congestion and packet latency from some heavy branches of the routing tree. If the WSN has a balanced tree, the energy consumption of nodes at a certain level of the tree may be distributed evenly and the lifetime of the network may be prolonged. In this thesis, the performance evaluation of our solutions is verified through extensive simulations. Our future plan is to have a real test-bed implementation, where we are going to implement ER-MAC in real sensor nodes using either the Contiki operating system [1] or TinyOS [3].

Our works on topology planning algorithms currently emphasise the fault-tolerant aspects of WSN topologies, including the availability of alternate paths and path length constraints. While these are the most important factors for WSN survivability, we will also include network capacity requirements and lifetime expectation in our algorithm designs. In this thesis, the GRASP-ARP topologies can guarantee robustness against multiple failures because Counting-Paths can find $k \geq 2$ disjoint paths. Our immediate future work will be a complete analysis of the dynamic programming variant of Counting-Paths. We also consider GRASP-ABP and GRASP-MSRP topologies that can only guarantee robustness against one single failure, because our current l-CRC index only checks the availability of one length-bounded alternate path. In our future work, we will try to extend the l-CRC index by taking into account the availability of more than one alternate path. Consequently, the GRASP-ABP and GRASP-MSRP topologies will be designed to guarantee robustness against multiple failures. Moreover, we currently assume point-based failures, where the failed devices are scattered in the network. In our

future work, we will consider area-based failures, where failed devices are in close proximity to each other. In addition, this thesis assumes predetermined positions of sensor nodes in the topologies. Our future work will also include sensor node deployment, where we will not only preserve connectivity and survivability of the network, but also maintain the coverage requirements. For the coverage, a particular WSN application may require that every point in the monitoring area is sensed by at least $k \geq 1$ sensor nodes.

Appendix A

Graph Model for WSN

A.1 Notations and Definitions

A WSN which consists of n sensor nodes can be modeled as a graph $G = (V, E)$, where V represents the set of vertices, and E is the set of edges (v, w) for $v, w \in V$ and vertex v can communicate by radio directly with vertex w. Some WSN protocols require bi-directional links between each pair of nodes to facilitate link level acknowledgement, which is critical for packet transmissions over unreliable wireless links. Therefore, for simplicity, we assume bi-directional links, where v and w are *adjacent* if they are within transmission range of each other. However, this assumption could be easily relaxed by specifying a more complex connectivity graph.

Let $v \in V$ be a vertex in G, we define *neighbourhood* $N(v)$ as the set of all vertices that are adjacent to v. Formally, $N(v) = \{w : w \in V, (v, w) \in E\}$. $H = (W, E{\downarrow_W})$ is an induced subgraph of $G = (V, E)$ if $W \subset V$ and $E{\downarrow_W}$ has exactly the edges that appear in G over the same vertex set (where $E{\downarrow_X}$ means a set of edges restricted to those that connect nodes in X). A *path* of length t between two vertices v and w is a sequence of vertices $v = v_0, v_1, \ldots, v_t = w$, such that v_i and v_{i+1} are adjacent for each i. A path from a vertex v to a set of vertices W is simply a path from v

to any vertex $w \in W$. Two vertices are *connected* if there is a path between them. A graph is connected if every pair of vertices is connected. A *cutset* is a set $C \subset V$ such that $(V-C, E\downarrow_{V-C})$ is disconnected. A graph is *k-connected* if it has no cutset of size less than k. Two paths P and Q from v to w are *vertex-disjoint* if they have no vertices in common except for v and w.

Below is the theorem for graph k-connectivity by Menger (1927) as cited from [36]. Before we present the theorem, we give the following definitions. Given sets $A, B \subseteq V$, we call $P = v_0, \ldots, v_t$ an $A-B$ *path* if $V(P) \cap A = \{v_0\}$ and $V(P) \cap B = \{v_t\}$. An $A-B$ *cutset* is a cutset that separates the sets A and B in G. Let k be the minimum size of an $A-B$ cutset. Clearly, G cannot contain more than k disjoint $A-B$ paths.

Theorem A.1. (Menger, 1927) *Let $G=(V,E)$ be a graph and $A, B \subseteq V$. Then the minimum number of vertices separating A from B is equal to the maximum number of disjoint $A-B$ paths in G.*

The WSN topology is an undirected graph and for simplicity, we assume that the graph is connected. ω is a *weight function* of an edge. For an edge $(v,w) \in E$, we define $\omega(v,w) = 1$ for *unweighted* graphs and $\omega(v,w) > 0$ for *weighted* graphs. For $u, w \in V$, $d(u,w)$ denotes the shortest path *distance* between u and w. By convention, $d(u,w) = \infty$ if w is unreachable from u and $d(u,u) = 0$. We denote $d_v(u,w)$ to represent the distance of the shortest path from u to w which does not visit v. Let σ_{st} denotes the number of shortest paths between s and t and let $\sigma_{st}(v)$ be the number of shortest paths between s and t that passing through some vertex v other than s and t. By convention, if $s=t$, $\sigma_{st}=1$ and if $v \in \{s,t\}$, $\sigma_{st}(v)=0$. Let $l_{\max}$ denote the maximum acceptable path length, we say that a vertex v is *k-covered* by a set of k vertices W if $d(v,w) \leq l_{\max}; \forall w \in W$. If $k=1$, we simply say v is *single-covered*. If $k=2$, v is *double-covered*.

In a WSN with a data sink, the routing paths from all sensor nodes to the sink form a *rooted tree*, where the sink is the root of the tree. Any vertex w on a path

Algorithm 17: Ford-Fulkerson

Input : G, s, t
Output: *flow*

```
 1: for each (v, w) ∈ E(G) do
 2:     flow(v, w) ← 0
 3:     flow(w, v) ← 0
 4: end for
 5: while there exist a path P from s to t in the residual network G_res do
 6:     capacity_res(P) ← min{capacity_res(v, w) : (v, w) is in P}
 7:     for each (v, w) in P do
 8:         flow(v, w) ← flow(v, w) + capacity_res(P)
 9:         flow(w, v) ← −flow(v, w)
10:     end for
11: end while
12: return flow
```

from a vertex v to the root is an *ancestor* of v. If w is an ancestor of v, then v is a *descendant* of w. The *subtree rooted at* v is the tree induced by descendants of v rooted at v. In a tree, v is the *parent* of w and w is the *child* of v if an edge (v, w) exists with $d(v, Sink) < d(w, Sink)$. For WSNs with multiple sinks, a well-known approach is by adding a *supersink* as an imaginary vertex that has connection to the original sinks [20]. By doing this, we reduce the problem of multiple sinks to the problem of single sink. Two vertices with the same parent are *siblings*. A vertex with no children is a *leaf* and a vertex with children is a *non-leaf*.

A.2 The Ford-Fulkerson Algorithm

We present the Ford-Fulkerson method in Algorithm 17 as cited from [34]. This method is iterative. From line 1 to 4, we initialise *flow* to 0. The loop from line 5 to 11 repeatedly finds an augmenting path P in the residual network G_{res} and augments *flow* along P by the residual capacity $capacity_{\text{res}}(P)$. The residual network G_{res} is the network with residual capacity $capacity_{\text{res}}(v, w) = capacity(v, w) - flow(v, w)$. When no more augmenting paths exist, the flow f is a maximum flow.

Appendix B

The Disjoint Path Algorithms

B.1 Shortest Vertex-Disjoint Paths with Modified Dijkstra by Bhandari

In [19], Bhandari gives a variant of the original Dijkstra algorithm. It takes as input a graph $G=(V,E)$, a weight function $\omega : E \rightarrow \mathbb{R}$ associated with its edges, a source s, and a destination t. Let $d(v)$ denotes the distance of vertex v from s, $\pi(v)$ denotes v's parent on the shortest path, and $N(v)$ is the set of v's neighbours. The pseudocode is given in Algorithm 18. In each iteration, it searches for a vertex in the set S with the least path length. It terminates when the selected vertex is the destination.

The original Dijkstra algorithm only works for the case in which all edge weights are non-negative. In this algorithm, when a vertex with the least path length is selected, the shortest path to that vertex has been found and no further scanning from any other vertices in the graph can update its distance to the source. On the other hand, Modified Dijkstra can handle negative directed edges. The modification allows that a previously selected vertex can be rescanned and so its distance to the source vertex can be updated.

Algorithm 18: Modified Dijkstra

Input : G, ω, s, t
Output: d, π

1: $d(s) \leftarrow 0, \quad \pi(s) \leftarrow$ **NIL**
2: **for** all $v \in V \setminus \{s\}$ **do**
3: **if** $v \in N(s)$ **then**
4: $d(v) \leftarrow \omega(s, v), \quad \pi(v) \leftarrow s$
5: **else**
6: $d(v) \leftarrow \infty, \quad \pi(v) \leftarrow$ **NIL**
7: **end if**
8: **end for**
9: $S \leftarrow N(s)$
10: **while** $S \neq \emptyset$ **do**
11: Find $v \in S$ such that $d(v) = \min\{d(u)\}, \forall u \in S$
12: $S \leftarrow S \setminus \{v\}$
13: **if** $v \neq t$ **then**
14: **for** all $u \in N(v)$ **do**
15: **if** $d(u) > d(v) + \omega(v, u)$ **then**
16: $d(u) \leftarrow d(v) + \omega(v, u), \quad \pi(u) \leftarrow v$
17: $S \leftarrow S \cup \{u\}$
18: **end if**
19: **end for**
20: **else**
21: **return** d, π
22: **end if**
23: **end while**

Algorithm 19 is proposed by Bhandari to solve the problem of finding single source – single sink shortest vertex-disjoint paths. It takes as input the original graph $G = (V, E)$, a weight function $\omega : E \rightarrow \mathbb{R}$ associated with its edges, a source s, a destination t, and the number of disjoint paths sought k. In each iteration, it finds the shortest path using the Modified Dijkstra algorithm.

B.2 Fast Pathfinding by Torrieri

The original Fast Pathfinding algorithm [113] finds all possible disjoint paths from length = 1 to the maximum acceptable path length $l_{\max}$ between two vertices. Instead of finding all possible disjoint paths, we slightly modify this algorithm to only find the shortest k disjoint paths.

Algorithm 19: Modified Dijkstra for Single Source – Single Sink Disjoint Paths

Input : G, ω, s, t, k
Output: $P_i, \forall i = 1, \ldots, k$

```
for i ← 1 to k do
    if i > 1 then
        Replace each edge on the shortest paths with a negative edge directed
        towards s
        Split each vertex on the shortest paths except s and t into original vertex
        and primed vertex, which are joined by a directed edge of length zero
        from the primed vertex to the original vertex towards s
        Replace each external edge connected to the vertex on the shortest paths
        by two oppositely directed edges of the same length: one edge terminates
        on the original vertex, while another edge originates from the primed
        vertex
    end if
    Find the shortest path P_i using the Modified Dijkstra algorithm
    if i > 1 then
        Remove the zero length edges, merge the primed and original vertices
        Replace the directed edges with their original edges
        Remove overlapping edges of the paths to get the shortest disjoint paths
    end if
end for
return P_i, ∀i = 1, ..., k
```

The pseudocode for the Fast Pathfinding algorithm is presented in Algorithm 20. It takes as input an $n \times n$ adjacency matrix G of a graph of n vertices, a source s, a destination t, the number of disjoint paths sought k, and $l_{\max}$. In the adjacency matrix, $G(a, b) = 1$ if there is an edge from vertex a to vertex b and $G(a, b) = 0$ if there is not. Fast Pathfinding starts by finding the shortest path of length $= 1$, i.e. there is no intermediate vertices in the path. Then, it gradually increases the number of intermediate vertices. Let G' be the reduced adjacency matrix of G. In each iteration, the algorithm selects one shortest path P, removes the intermediate vertices in P from further use by zeroing the rows and the columns of the intermediate vertices in G', and then selects the next shortest path using only the remaining vertices. If two or more remaining paths of length l are the shortest, one of them is chosen arbitrarily.

Algorithm 20: Fast Pathfinding

Input : $G, s, t, k, l_{\max}$
Output: $P_i, \forall i = 1, \ldots, k$

$G' \leftarrow G$, $numDisjointPath \leftarrow 1$, $numIntermediateVertices \leftarrow 0$
while $numDisjointPath < k$ **and** $numIntermediateVertices < l_{\max}$ **do**
  if $numIntermediateVertices = 0$ **then**
    /* Path of length = 1 */
    if $G(s,t) = 1$ **then**
      $G'(s,t) \leftarrow 0$
      Zeroing column s and row t in G'
      $P_{numDisjointPath} \leftarrow \{s, t\}$
      $numDisjointPath \leftarrow numDisjointPath + 1$
    end if
  else
    /* Path of length > 1 */
    $idx_{\mathrm{T}} \leftarrow (numIntermediateVertices + 2)/2$
    $idx_{\mathrm{S}} \leftarrow (numIntermediateVertices + 1)/2$
    $T_0 \leftarrow \{s\}$, $S_0 \leftarrow \{t\}$
    for $i \leftarrow 1$ to idx_{T} **do**
      $T_i \leftarrow$ non-zero vertices of row v in G', $\forall v \in T_{i-1}$
    end for
    for $i \leftarrow 1$ to idx_{S} **do**
      $S_i \leftarrow$ non-zero vertices of column v in G', $\forall v \in S_{i-1}$
    end for
    $TS \leftarrow T_{idx_{\mathrm{T}}} \cap S_{idx_{\mathrm{S}}}$
    for $i \leftarrow 1$ to $|TS|$ **do**
      /* Backward search to s */
      $idx \leftarrow idx_{\mathrm{T}}$
      $P_{numDisjointPath}(idx) \leftarrow TS(i)$, $idx \leftarrow idx - 1$
      for $j \leftarrow idx_{\mathrm{T}} - 1$ to 0 **do**
        Select $v \in T_j$ where $G(v, P_{numDisjointPath}(idx+1)) = 1$
        $P_{numDisjointPath}(idx) \leftarrow v$, $idx \leftarrow idx - 1$
      end for
      /* Forward search to t */
      $idx \leftarrow idx_{\mathrm{T}} + 1$
      for $j \leftarrow idx_{\mathrm{S}} - 1$ to 0 **do**
        Select $v \in S_j$ where $G(P_{numDisjointPath}(idx-1), v) = 1$
        $P_{numDisjointPath}(idx) \leftarrow v$, $idx \leftarrow idx + 1$
      end for
      Zeroing column v and row v in G', $\forall$ intermediate v in $P_{numDisjointPath}$
      $numDisjointPath \leftarrow numDisjointPath + 1$
    end for
  end if
  $numIntermediateVertices \leftarrow numIntermediateVertices + 1$
end while
return $P_i, \forall i = 1, \ldots, k$

B.3 Maximum Paths by Torrieri

Similar to Fast Pathfinding, the original Maximum Paths algorithm [113] finds all possible disjoint paths from length = 1 to the maximum acceptable path length $l_{\max}$ between two vertices. The slight modification of this algorithm to only find the shortest k disjoint paths is presented in Algorithm 21. It takes as input an adjacency matrix G, a source s, a destination t, the number of disjoint paths sought k, and $l_{\max}$. Maximum Paths differs from Fast Pathfinding in that the number of excluded paths must be determined. If two or more remaining paths of length l are the shortest and they exclude the fewest other paths of length l, then one of the remaining paths is chosen arbitrarily.

Algorithm 21: Maximum Paths

Input : $G, s, t, k, l_{\max}$
Output: $P_i, \forall i = 1, \ldots, k$

$G' \leftarrow G$, $numDisjointPath \leftarrow 1$, $numIntermediateVertices \leftarrow 0$
while $numDisjointPath < k$ **and** $numIntermediateVertices < l_{\max}$ **do**
  if $numIntermediateVertices = 0$ **then**
    /* Path of length = 1 */
    if $G(s,t) = 1$ **then**
      $G'(s,t) \leftarrow 0$
      Zeroing column s and row t in G'
      $P_{numDisjointPath} \leftarrow \{s,t\}$
      $numDisjointPath \leftarrow numDisjointPath + 1$
    end if
  else
    /* Path of length > 1 */
    $idx_{\mathrm{T}} \leftarrow (numIntermediateVertices + 2)/2$
    $idx_{\mathrm{S}} \leftarrow (numIntermediateVertices + 1)/2$
    $T_0^1 \leftarrow \{s\}$, $S_0^1 \leftarrow \{t\}$
    for $i \leftarrow 1$ to idx_{T} **do**
      $n \leftarrow 1$
      for $j \leftarrow 1$ to $|T_{i-1}|$ **do**
        for each non-zero vertex v of row w in G', w is the last vertex of T_{i-1}^j **do**
          $T_i^n \leftarrow T_{i-1}^j \cup \{v\}$, $n \leftarrow n+1$
        end for
      end for
    end for
    for $i \leftarrow 1$ to idx_{S} **do**
      $n \leftarrow 1$
      for $j \leftarrow 1$ to $|S_{i-1}|$ **do**
        for each non-zero vertex v of column w in G', w is the first vertex of S_{i-1}^j **do**
          $S_i^n \leftarrow \{v\} \cup S_{i-1}^j$, $n \leftarrow n+1$
        end for
      end for
    end for
    $n \leftarrow 1$
    for $i \leftarrow 1$ to $|T_{idx_{\mathrm{T}}}|$ **do**
      for $j \leftarrow 1$ to $|S_{idx_{\mathrm{S}}}|$ **do**
        if last vertex of $T_{idx_{\mathrm{T}}}^i$ = first vertex of $S_{idx_{\mathrm{S}}}^j$ **then**
          $TS_n \leftarrow T_{idx_{\mathrm{T}}}^i \cup S_{idx_{\mathrm{S}}}^j$, $n \leftarrow n+1$
        end if
      end for
    end for
    do
      Find TS_i that excludes the fewest other paths
      $P_{numDisjointPath} \leftarrow TS_i$
      Remove TS_i from TS
      Zeroing column v and row v in G', $\forall$ intermediate v in $P_{numDisjointPath}$
      $numDisjointPath \leftarrow numDisjointPath + 1$
    while $|TS| > 0$
  end if
  $numIntermediateVertices \leftarrow numIntermediateVertices + 1$
end while
return $P_i, \forall i = 1, \ldots, k$

Appendix C

The Partial k-Connectivity-Repair Algorithm for Relay Placement

The k-Connectivity-Repair algorithm was originally proposed by Bredin *et al.* [26] for full fault-tolerant relay placement. It was modified by Pu *et al.* [90] for partial fault-tolerance. The pseudocode presented in Algorithm 22 is the Partial k-Connectivity-Repair algorithm (K-CONN-REPAIR) from [90] with some modifications to work in constrained deployment locations.

K-CONN-REPAIR takes as input the original graph $G=(T, E)$ where T is the set of sensors, the set of candidate relays A, the number of disjoint paths sought k, and the transmission range of a sensor r. It starts by computing a weighted complete graph. The weight of an edge is one less than the Euclidean distance between two sensors. It is roughly equivalent to the number of relays needed to connect the two sensors. After that, K-CONN-REPAIR finds an approximate minimum-weight vertex k-connected subgraph by repeatedly adding edges in increasing order of weight until the subgraph is k-connected. If the subgraph is k-connected, it repeatedly attempts to remove edges in decreasing order of weight, but putting the edge back if it is important for k-connectivity. The k-connectivity is checked using a maximum network-flow-based checking algorithm [86, 95]. Finally, for each

Algorithm 22: K-CONN-REPAIR

Input : G, A, k, r
Output: R

/* Compute a weighted complete graph */
1: $G' \leftarrow (T, E'),\ E' \leftarrow \{(v, w) \mid v, w \in T, v \neq w\}$
2: $\omega(v, w) \leftarrow \lceil\, distance(v, w) \,/\, r \,\rceil - 1,\ \forall v, w \in T$
/* Compute an approximate minimum-weight */
/* k-connected spanning subgraph */
3: $G'' \leftarrow (T, \emptyset),\ E'' \leftarrow \{(v, w) \mid v, w \in T, v \neq w\}$
4: **for** each $(v, w) \in E''$ in increasing order of $\omega(v, w)$ **do**
5: $E(G'') \leftarrow E(G'') \cup \{(v, w)\}$
6: **if** G'' is k-connected **then**
7: **break**
8: **end if**
9: **end for**
10: **for** each $(v, w) \in E(G'')$ in decreasing order of $\omega(v, w)$ **do**
11: $G''' \leftarrow (T, E(G'') \setminus \{(v, w)\})$
12: **if** G''' is k-connected **then**
13: $G'' \leftarrow G'''$
14: **end if**
15: **end for**
/* Deploy relays */
16: $H \leftarrow (T \cup A, E{\downarrow_{T \cup A}})$
17: **for** each $(v, w) \in E(G'')$ **do**
18: Find the shortest relay path from v to w in H
19: $R \leftarrow R \cup$ relays that appear on the shortest relay path
20: If we need to find more shortest relay paths originating from v, we need to temporarily remove the relays (from H) on the previously found shortest relay paths originating from v, so the paths are disjoint
21: **end for**
22: $H' \leftarrow (T \cup R, E{\downarrow_{T \cup R}})$
23: Try to remove relays in H' one by one but still preserving k-connectivity
24: **return** R

edge that appears in the subgraph, the algorithm places relays along the shortest relay path between the two endpoints of the edge. In a shortest relay path, the intermediate vertices are candidate relays. In order to obtain disjoint paths, we need to temporarily remove the relays on the previously found shortest relay paths originating from a sensor v before finding more shortest relay paths originating from v. When all relays are deployed, we try to remove relays one by one by still preserving k-connectivity.

In the maximum network-flow-based checking algorithm, an undirected graph G must be firstly converted into a directed graph by replacing each undirected edge in G with a pair of opposite directed edges. Then a directed graph G' is constructed from G as follows:

1. each vertex v in G is split into two vertices: the original vertex v and the primed vertex v',
2. for each vertex v in G, adds a directed edge (v', v) in G',
3. for each directed edge (v, w) in G, adds a directed edge (v, w') into G',
4. for each directed edge (w, v) in G, adds a directed edge (w, v') into G',
5. assigns each edge in G' a capacity of one.

The connectivity between two vertices v and w in G is equal to the maximum network flow between v and w' in G'. The graph connectivity is checked by calculating the connectivity between every pair of vertices.

Appendix D

The Multiple Sink and Relay Placement Algorithms

D.1 The Multiple Sink Placement Algorithms

In this section, we will give the pseudocode for *Minimise the Number of Sinks for Fault-Tolerance* (MSFT) and *Cluster-Based Sampling for Multiple Sink Placement* (CBS-MSP). These two algorithms based on the well-known k-means clustering algorithm. It divides a network into clusters and finds the position of each sink, which is in the centre of a cluster.

D.1.1 Minimise the Number of Sinks for Fault-Tolerance (MSFT)

MSFT is similar to *Minimise the Number of Sinks for a Predefined Minimum Operation Period* (MSPOP) [85]. In MSPOP, the deployment locations of sinks are unconstrained and the objective is to place sinks one by one until a required lifetime is met. Unlike MSPOP, MSFT deploys sinks at candidate locations until the network is double-covered.

Algorithm 23: MSFT

Input : $G, T, A_{\mathrm{S}}, c, Distance_G, l_{\max}, max\_iterations$
Output: S^*

1: $best\_cost \leftarrow \infty$
2: **for** $i \leftarrow 1$ to $max\_iterations$ **do**
3: $n \leftarrow 2$
4: **do**
5: $S \leftarrow$ select n sinks from A_{S} randomly
6: **do**
7: $S' \leftarrow \emptyset$
8: **for** all $v \in S$ **do**
9: $Cluster(v) \leftarrow$ all vertices $\subset T$ that have v as the nearest sink
10: Find $w \in A_{\mathrm{S}}$ such that the mean distance from all vertices in $Cluster(v)$ to w is the smallest
11: $S' \leftarrow S' \cup \{w\}$
12: **end for**
13: $S \leftarrow S'$
14: **while** sinks can be moved
15: $H \leftarrow (T \cup S, E{\downarrow_{T \cup S}})$
16: Calculate $num\_uncovered$ in H using $Distance_G$ and $l_{\max}$
17: $n \leftarrow n+1$
18: **while** $num\_uncovered > 0$ **and** $n \leq |A_{\mathrm{S}}|$
19: **if** $\sum_{v \in S} c_v < best\_cost$ **then**
20: $S^* \leftarrow S, \quad best\_cost \leftarrow \sum_{v \in S} c_v$
21: **end if**
22: **end for**
23: **return** S^*

The performance of MSFT depends on the initial randomly selected sink locations, so more iterations give better results. We limit the number of iterations by *max_iterations*. The MSFT pseudocode is given in Algorithm 23. It takes as input the original graph $G = (T \cup A_S, E)$, the set T of sensors, the set A_S of candidate sinks, the cost function c, the pre-computed $Distance_G$ table, the maximum acceptable path length l_{max}, and *max_iterations*. In each iteration, it starts by trying to minimise the number of uncovered sensors using two sinks. The sinks are selected randomly from the candidate locations, then clusters are constructed, where each sensor joins its nearest sink's cluster. After the cluster formation, we find the best new position for a sink such that the mean distance from all sensors in the cluster to the sink is minimum. This process is repeated until all sinks cannot be moved. If there are some uncovered sensors, MSFT increases the number of required sinks. The end result of this algorithm is a set of sinks with the minimum total cost.

D.1.2 Cluster-Based Sampling for Multiple Sink Placement (CBS-MSP)

CBS-MSP modifies *Cluster-Based Sampling* (CBS) proposed in [28]. In CBS, the number of sinks is given as an input to the algorithm with an objective to minimise the total road distance from all nodes to the sinks, where each node is required to be double-covered. Unlike CBS, CBS-MSP minimises the number of deployed sinks and the deployment cost. We implement CBS-MSP using path length to represent distance between two nodes and also we have a path length limit.

The pseudocode for CBS-MSP is given in Algorithm 24. Its implementation is very similar to MSFT. The key differences between these two algorithms are in line 9 and 10, where MSFT constructs two clusters. The primary clusters of a sink v consists of sensors that have v as their nearest sink. The secondary cluster of v are sensors that have v as their second nearest sink. The best new position for a sink is sought where the mean distance from all sensors in both clusters to the sink is minimum.

Algorithm 24: CBS-MSP

Input : $G, T, A_{\mathrm{S}}, c, Distance_G, l_{\max}, max\_iterations$
Output: S^*

$best\_cost \leftarrow \infty$
for $i \leftarrow 1$ to $max\_iterations$ **do**
  $n \leftarrow 2$
  do
    $S \leftarrow$ select n sinks from A_{S} randomly
    do
      $S' \leftarrow \emptyset$
      for all $v \in S$ **do**
        $Primary\_Cluster(v) \leftarrow$ all vertices $\subset T$ that have v as the nearest sink,
        $Secondary\_Cluster(v) \leftarrow$ all vertices $\subset T \setminus Primary\_Cluster(v)$ that have v as the second nearest sink
        Find $w \in A_{\mathrm{S}}$ such that the mean distance from all vertices in $Primary\_Cluster(v) \cup Secondary\_Cluster(v)$ to w is the smallest
        $S' \leftarrow S' \cup \{w\}$
      end for
      $S \leftarrow S'$
    while sinks can be moved
    $H \leftarrow (T \cup S, E{\downarrow_{T \cup S}})$
    Calculate $num\_uncovered$ in H using $Distance_G$ and $l_{\max}$
    $n \leftarrow n+1$
  while $num\_uncovered > 0$ **and** $n \leq |A_{\mathrm{S}}|$
  if $\sum_{v \in S} c_v < best\_cost$ **then**
    $S^* \leftarrow S, \quad best\_cost \leftarrow \sum_{v \in S} c_v$
  end if
end for
return S^*

D.2 The Multiple Sink and Relay Placement Algorithms

In this section, we will give the pseudocode for *Minimise the Number of Sinks and Relays for Fault-Tolerance* (MSRFT) and *Cluster-Based Sampling for Multiple Sink and Relay Placement* (CBS-MSRP). Both algorithms iteratively find the best locations to deploy a given number of sinks, which is obtained by giving the number of required sinks as an input parameter to MSFT and CBS-MSP. Then, they utilise GRASP-MRP from Chapter 7 to deploy relays.

D.2.1 Minimise the Number of Sinks and Relays for Fault-Tolerance (MSRFT)

MSRFT extends MSFT to find the best locations to deploy sinks and uses GRASP-MRP to deploy relays until the network becomes double-covered and non-critical. Firstly, we modify MSFT to find the best locations for a given number of sinks. We call this modification *Find the Best Sink Locations for Fault-Tolerance* (BSLFT). The pseudocode for BSLFT is given in Algorithm 25. BSLFT differs from MSFT in that it takes the number of required sinks n as one of its input and the objective is to find the best locations for n sinks such that the number of uncovered sensors is minimised.

MSRFT is presented in Algorithm 26. It takes as input the original graph $G = (T \cup A_{\mathrm{R}} \cup A_{\mathrm{S}}, E)$, the set T of sensors, the set A_{R} of candidate relays, the set A_{S} of candidate sinks, the cost function c, the pre-computed $Distance_G$ table, the maximum acceptable path length $l_{\max}$, the number of iterations *max_iterations* for BSLFT, and the number of iterations *max_iterations_grasp* for GRASP-MRP. MSRFT starts by deploying two sinks using BSLFT and calls GRASP-MRP to deploy relays. It then gradually increases the number of sinks until the network becomes double-covered and non-critical.

Algorithm 25: BSLFT

Input : $G, T, A_{\mathrm{S}}, c, \mathit{Distance}_G, l_{\max}, \mathit{max\_iterations}, n$
Output: S^*

$\mathit{best\_value} \leftarrow \infty$
for $i \leftarrow 1$ to $\mathit{max\_iterations}$ **do**
  $S \leftarrow$ select n sinks from A_{S} randomly
  do
    $S' \leftarrow \emptyset$
    for all $v \in S$ **do**
      $\mathit{Cluster}(v) \leftarrow$ all vertices $\subset T$ that have v as the nearest sink
      Find $w \in A_{\mathrm{S}}$ such that the mean distance from all vertices in $\mathit{Cluster}(v)$ to w is the smallest
      $S' \leftarrow S' \cup \{w\}$
    end for
    $S \leftarrow S'$
  while sinks can be moved
  $H \leftarrow (T \cup S, E{\downarrow_{T \cup S}})$
  Calculate $\mathit{num\_uncovered}$ in H using $\mathit{Distance}_G$ and $l_{\max}$
  if $\mathit{num\_uncovered} < \mathit{best\_value}$ **then**
    $S^* \leftarrow S, \quad \mathit{best\_value} \leftarrow \mathit{num\_uncovered}$
  end if
end for
return S^*

Algorithm 26: MSRFT

Input : $G, T, A_{\mathrm{R}}, A_{\mathrm{S}}, c, \mathit{Distance}_G, l_{\max}, \mathit{max\_iterations}, \mathit{max\_iterations\_grasp}$
Output: R^*, S^*

$\mathit{best\_cost} \leftarrow \infty$
$n \leftarrow 2$
do
  $S \leftarrow \mathrm{BSLFT}(G, T, A_{\mathrm{S}}, c, \mathit{Distance}_G, l_{\max}, \mathit{max\_iterations}, n)$
  $H \leftarrow (T \cup A_{\mathrm{R}} \cup S, E{\downarrow_{T \cup A_{\mathrm{R}} \cup S}})$
  Calculate $\mathit{num\_uncovered}$ and $\mathit{num\_critical}$ in H using $\mathit{Distance}_G$ and $l_{\max}$
  if $\mathit{num\_uncovered} = 0$ **and** $\mathit{num\_critical} = 0$ **then**
    $R \leftarrow \mathrm{GRASP\text{-}MRP}(G, T, S, A_{\mathrm{R}}, l_{\max}, \mathit{max\_iterations\_grasp})$
    if $\sum_{v \in R \cup S} c_v < \mathit{best\_cost}$ **then**
      $R^* \leftarrow R, \quad S^* \leftarrow S, \quad \mathit{best\_cost} \leftarrow \sum_{v \in R \cup S} c_v$
    end if
  end if
  $n \leftarrow n + 1$
while $\mathit{num\_uncovered} > 0$ **and** $n \leq |A_{\mathrm{S}}|$
return R^*, S^*

Algorithm 27: CBS-BSL

Input : $G, T, A_{\mathrm{S}}, c, Distance_G, l_{\max}, max\_iterations, n$
Output: S^*

1: $best\_value \leftarrow \infty$
2: **for** $i \leftarrow 1$ to $max\_iterations$ **do**
3: $S \leftarrow$ select n sinks from A_{S} randomly
4: **do**
5: $S' \leftarrow \emptyset$
6: **for** all $v \in S$ **do**
7: $Primary\_Cluster(v) \leftarrow$ all vertices $\subset T$ that have v as the nearest sink, $Secondary\_Cluster(v) \leftarrow$ all vertices $\subset T \setminus Primary\_Cluster(v)$ that have v as the second nearest sink
8: Find $w \in A_{\mathrm{S}}$ such that the mean distance from all vertices in $Primary\_Cluster(v) \cup Secondary\_Cluster(v)$ to w is the smallest
9: $S' \leftarrow S' \cup \{w\}$
10: **end for**
11: $S \leftarrow S'$
12: **while** sinks can be moved
13: $H \leftarrow (T \cup S, E\downarrow_{T \cup S})$
14: Calculate $num\_uncovered$ in H using $Distance_G$ and $l_{\max}$
15: **if** $num\_uncovered < best\_value$ **then**
16: $S^* \leftarrow S, \quad best\_value \leftarrow num\_uncovered$
17: **end if**
18: **end for**
19: **return** S^*

D.2.2 Cluster-Based Sampling for Multiple Sink and Relay Placement (CBS-MSRP)

CBS-MSRP is similar to MSRFT, but it extends CBS-MSP. The modification of CBS-MSP to find the best locations for a given number of sinks is given in Algorithm 27, which we call *Cluster-Based Sampling for Finding the Best Sink Locations* (CBS-BSL). CBS-BSL is different from CBS-MSP because it takes the number of required sinks n as one of its input and the objective is to find the best locations for n sinks to minimise the number of uncovered sensors. CBS-MSRP is presented in Algorithm 28. Its difference to MSRFT is in line 4, i.e. instead of calling BSLFT, it calls CBS-BSL.

Algorithm 28: CBS-MSRP

Input : $G, T, A_{\mathrm{R}}, A_{\mathrm{S}}, c, \mathit{Distance}_G, l_{\max}, \mathit{max\_iterations}, \mathit{max\_iterations\_grasp}$
Output: R^*, S^*

$\mathit{best\_cost} \leftarrow \infty$
$n \leftarrow 2$
do
  $S \leftarrow$ CBS-BSL$(G, T, A_{\mathrm{S}}, c,\ \mathit{Distance}_G, l_{\max},\ \mathit{max_iterations},\ n)$
  $H \leftarrow (T \cup A_{\mathrm{R}} \cup S, E{\downarrow_{T \cup A_{\mathrm{R}} \cup S}})$
  Calculate *num_uncovered* and *num_critical* in H using $\mathit{Distance}_G$ and $l_{\max}$
  if $\mathit{num\_uncovered} = 0$ **and** $\mathit{num\_critical} = 0$ **then**
    $R \leftarrow$ GRASP-MRP$(G, T, S, A_{\mathrm{R}}, l_{\max},\ \mathit{max_iterations_grasp})$
    if $\sum_{v \in R \cup S} c_v < \mathit{best\_cost}$ **then**
      $R^* \leftarrow R, \quad S^* \leftarrow S, \quad \mathit{best\_cost} \leftarrow \sum_{v \in R \cup S} c_v$
    end if
  end if
  $n \leftarrow n + 1$
while $\mathit{num\_uncovered} > 0$ **and** $n \leq |A_{\mathrm{S}}|$
return R^*, S^*

Bibliography

[1] The Contiki Operating System. Available at `http://www.contiki-os.org` [2 August 2012].

[2] The Network Simulator - ns-2. Available at `http://www.isi.edu/nsnam/ns/` [30 April 2010].

[3] TinyOS. Available at `http://www.tinyos.net` [2 August 2012].

[4] Tmote Sky Datasheet. Available at `http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf` [30 April 2010].

[5] Z-MAC: Hybrid MAC for Wireless Sensor Networks. Available at `http://www4.ncsu.edu/~rhee/export/zmac/software/zmac/zmac.htm` [30 April 2010].

[6] M. Ahlberg, V. Vlassov and T. Yasui. Router Placement in Wireless Sensor Network. Technical Report KTH/ICT/ECS, Royal Institute of Technology (KTH), Stockholm, Sweden, 2006.

[7] G. Ahn, E. Miluzzo, A. T. Campbell, S. G. Hong and F. Cuomo. Funneling-MAC: A Localized, Sink-Oriented MAC for Boosting Fidelity in Sensor Networks. In *Proc. 4th Int'l Conf. Embedded Networked Sensor Systems (SenSys'06)*, pages 293–306, Nov. 2006.

[8] K. Akkaya and M. Younis. COLA: A Coverage and Latency aware Actor Placement for Wireless Sensor and Actor Networks. In *Proc. IEEE Vehicular Technology Conference (VTC'06)*, Sept. 2006.

[9] K. Akkaya and M. F. Younis. An Energy-Aware QoS Routing Protocol for Wireless Sensor Networks. In *Proc. 23rd IEEE Intl Conf. Distributed Computing Systems (ICDCS'03)*, pages 710–715, May 2003.

[10] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, Volume 38, Number 4, pages 393–422, Mar. 2002.

[11] I. F. Akyildiz and M. C. Vuran (editors). *Wireless Sensor Networks*. John Wiley & Sons, Inc., 2010.

[12] J. N. Al-Karaki and A. E. Kamal. Routing Techniques in Wireless Sensor Networks: A Survey. *IEEE Wireless Communications*, Volume 11, Number 6, pages 6–28, Dec. 2004.

[13] T. Andersen and S. Tirthapura. Wireless Sensor Deployment for 3D Coverage with Constraints. In *Proc. 6th Int'l Conf. Networked Sensing Systems (INSS'09)*, Jun. 2009.

[14] B. Aoun, R. Boutaba, Y. Iraqi and G. Kenward. Gateway Placement Optimisation in Wireless Mesh Networks with QoS Constraints. *IEEE Journal on Selected Areas in Communications*, Volume 24, Number 11, pages 2127–2136, Nov. 2006.

[15] E. A. Basha, S. Ravela and D. Rus. Model-Based Monitoring for Early Warning Flood Detection. In *Proc. 6th ACM Conf. Embedded Networked Sensor Systems (SenSys'08)*, pages 295–308, Nov. 2008.

[16] A. Bavelas. A Mathematical Model for Group Structure. *Human Organizations*, Volume 7, pages 16–30, 1948.

[17] Y. Bejerano. Efficient Integration of Multihop Wireless and Wired Networks with QoS Constraints. *IEEE/ACM Transactions on Networking*, Volume 12, Number 6, pages 1064–1078, Dec. 2004.

[18] J. Beutel, K. Römer, M. Ringwald and M. Woehrle. Deployment Techniques for Wireless Sensor Networks. In G. Ferrari (editor), *Sensor Networks: Where Theory Meets Practice*, pages 219–248. Springer, 2009.

[19] R. Bhandari. Optimal Physical Diversity Algorithms and Survivable Networks. In *Proc. 2nd IEEE Symp. Computers and Communications (ISCC'97)*, pages 433–441, Jul. 1997.

[20] R. Bhandari. *Survivable Networks: Algorithm for Diverse Routing*. Kluwer Academic Publishers, 1999.

[21] S. Binato and G. C. Oliveira. A Reactive GRASP for Transmission Network Expansion Planning. In C. C. Ribeiro and P. Hansen (editors), *Essays and Surveys in Metaheuristics*, pages 81–100. Kluwer Academic Publishers, 2002.

[22] A. Bogdanov, E. Maneva and S. Riesenfeld. Power-aware Base Station Positioning for Sensor Networks. In *Proc. 23rd Int'l Ann. Joint Conf. IEEE Computer and Communications Societies (INFOCOM'04)*, pages 575–585, Mar. 2004.

[23] A. Boukerche, X. Cheng and J. Linus. A Performance Evaluation of a Novel Energy-Aware Data-Centric Routing Algorithm in Wireless Sensor Networks. *Wireless Networks*, Volume 11, Number 5, pages 619–635, Sept. 2005.

[24] A. Boukerche, R. W. N Pazzi and R. B. Araujo. Fault-Tolerant Wireless Sensor Network Routing Protocols for the Supervision of Contex-Aware Physical Environments. *Journal of Parallel and Distributed Computing*, Volume 66, Number 4, pages 586–599, Apr. 2006.

[25] U. Brandes. On Variants of Shortest-Path Betweenness Centrality and Their Generic Computation. *Social Networks*, Volume 30, Number 2, pages 136–145, May 2008.

[26] J. L. Bredin, E. D. Demaine, M. Hajiaghayi and D. Rus. Deploying Sensor Networks with Guaranteed Capacity and Fault Tolerance. In *Proc. 6th ACM Int'l Symp. Mobile Ad Hoc Networking and Computing (MobiHoc'05)*, pages 309–319, May 2005.

[27] M. Buettner, G. V. Yee, E. Anderson and R. Han. X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. In *Proc. 4th Int'l Conf. Embedded Networked Sensor Systems (SenSys'06)*, pages 307–320, Nov. 2006.

[28] H. Cambazard, D. Mehta, B. O'Sullivan, L. Quesada, M. Ruffini, D. Payne and L. Doyle. A Combinatorial Optimisation Approach to the Design of Dual-Parented Long-Reach Passive Optical Networks. In *Proc. 23rd IEEE Int'l Conf. Tools with Artificial Intelligence (ICTAI'11)*, pages 785–792, Nov. 2011.

[29] M. Ceriotti, M. Corra, L. D'Orazio, R. Doriguzzi, D. Facchin, S. Guna, G. P. Jesi, R. Lo Cigno, L. Mottola, A. L. Murphy, M. Pescalli, G. P. Picco, D. Pregnolato and C. Torghele. Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels. In *Proc. 10th ACM/IEEE Int'l Conf. Information Processing in Sensor Networks (IPSN'11)*, pages 187–198, Apr. 2011.

[30] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta and P. Zanon. Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment. In *Proc. 8th ACM/IEEE Int'l Conf. Information Processing in Sensor Networks (IPSN'09)*, pages 277–288, Apr. 2009.

[31] S. Chen, H. Bao, X. Zeng and Y. Yang. A Fire Detecting Method based on Multi-Sensor Data Fusion. In *Proc. IEEE Int'l Conf. Systems, Man and Cybernetics (SMC'03)*, pages 3775–3780, Oct. 2003.

[32] K. K. Chintalapudi. i-MAC - A MAC that Learns. In *Proc. 9th ACM/IEEE Int'l Conf. Information Processing in Sensor Networks (IPSN'10)*, pages 315–326, Apr. 2010.

[33] O. Chipara, Z. He, G. Xing, Q. Chen, X. Wang, C. Lu, J. Stankovic and T. Abdelzaher. Real-time Power-Aware Routing in Wireless Sensor Networks. In *Proc. 14th IEEE Workshop Quality of Service (IWQoS'06)*, pages 83–92, Jun. 2006.

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms, 2nd edition.* The MIT Press, 2001.

[35] H. Delmaire, J. A. Díaz, E. Fernández and M. Ortega. Reactive GRASP and Tabu Search Based Heuristics for the Single Source Capacitated Plant Location Problem. *INFOR*, Volume 47, pages 194–225, 1999.

[36] R. Diestel. *Graduate Texts in Mathematics: Graph Theory, 3rd edition.* Springer, 2005.

[37] R. S. Dubey, R. Choubey and A. Dubey. Mitigating Congestion Aware Routing Protocol in Wireless Sensor Networks. *Int'l Journal Engineering Science and Technology*, Volume 2, Number 12, pages 7395–7400, 2010.

[38] E. Egea-Lopez, J. Vales-Alonso, A. S. Martinez-Sala, J. Garcia-Haro, P. Pavon-Marino and M. V. Bueno-Delgado. A Real-Time MAC Protocol for Wireless Sensor Networks: Virtual TDMA for Sensors (VTS). In W. Grass, B. Sick and K. Waldschmidt (editors), *Proc. 19th Int'l Conf. Architecture of Computing Systems (ARCS'006)*, Volume 3894 LNCS, pages 382–396, Mar. 2006.

[39] A. El-Hoiydi and J. D. Decotignie. WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks. In S. E. Nikoletseas and J. D. P. Rolim (editors), *Proc. 1st Int'l Workshop Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS'04)*, Volume 3121 LNCS, pages 18–31, Jul. 2004.

[40] J. Elson and D. Estrin. Time Synchronization for Wireless Sensor Networks. In *Proc. 15th IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS'01)*, Apr. 2001.

[41] T. A. Feo and M. G. C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, Volume 8, pages 67–71, 1989.

[42] T. A. Feo and M. G. C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, Volume 6, pages 109–133, 1995.

[43] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton, University Press, 1962.

[44] L.C. Freeman. Centrality in Social Networks Conceptual Clarification. *Social Networks*, Volume 1, Number 3, pages 215–239, 1979.

[45] S. Ganeriwal and R. Kumar M. B. Srivastava. Timing-sync Protocol for Sensor Networks. In *Proc. 1st Int'l Conf. Embedded Networked Sensor Systems (SenSys'03)*, pages 138–149, Nov. 2003.

[46] C. F. García-Hernández, P. H. Ibargüengoytia-González, J. Garca-Hernández and J. A. Pérez-Daz. Wireless Sensor Networks and Applications: A Survey. *International Journal of Computer Science and Network Security (IJCSNS)*, Volume 7, Number 3, pages 264–273, Mar. 2007.

[47] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss and P. Levis. Collection Tree Protocol. In *Proc. 7th ACM Conf. Embedded Networked Sensor Systems (SenSys'09)*, Nov. 2009.

[48] H. Gong, M. Liu, Y. Mao, L. Chen and L. Xie. Traffic Adaptive MAC Protocol for Wireless Sensor Network. In X. Lu and W. Zhao (editors), *Proc. 3rd Int'l Conf. Computer Network and Mobile Computing (ICCNMC'05)*, Volume 3619 LNCS, pages 1134–1143, Aug. 2005.

[49] D. T. Gottuk, M. J. Peatross, R. J. Roby and C. L. Beyler. Advanced Fire Detection Using Multi-Signature Alarm Algorithms. *Fire Safety Journal*, Volume 37, pages 381–394, 2002.

[50] G. P. Halkes and K. G. Langendoen. Crankshaft: An Energy-Efficient MAC-Protocol for Dense Wireless Sensor Networks. In K. Langendoen and T. Voigt (editors), *Proc. 4th European Conf. Wireless Sensor Networks (EWSN'07)*, Volume 4373 LNCS, pages 228–244, Jan. 2007.

[51] X. Han, X. Cao, E. L. Lloyd and C. C. Shen. Fault-tolerant Relay Node Placement in Heterogeneous Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, Volume 9, Number 5, pages 643–656, May 2010.

[52] B. Hao, J. Tang and G. Xue. Fault-Tolerant Relay Node Placement in Wireless Sensor Networks: Formulation and Approximation. In *Proc. Workshop High Performance Switching and Routing (HPSR'04)*, pages 246–250, Apr. 2004.

[53] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu abd G. Zhou, J. Hui and B. Krogh. VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance. *ACM Transactions on Sensor Networks*, Volume 2, Number 1, pages 1–38, Feb. 2006.

[54] W. R. Heinzelman, A. P. Chandrakasan and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proc. 33rd Hawaii Int'l Conf. System Sciences (HICSS'00)*, Jan. 2000.

[55] R. Hou and H. Shi. A Localized Algorithm for Finding Disjoint Paths in Wireless Sensor Networks. *IEEE Communications Letters*, Volume 10, Number 12, pages 807–809, Dec. 2006.

[56] P. Hurni and T. Braun. MaxMAC: A Maximally Traffic-Adaptive MAC Protocol for Wireless Sensor Networks. In J. S Silva, B. Krishnamachari and F. Boavida (editors), *Proc. 7th European Conf. Wireless Sensor Networks (EWSN'10)*, Volume 5970 LNCS, pages 289–305, Feb. 2010.

[57] T. Issariyakul and E. Hossain (editors). *Introduction to Network Simulator NS2*. Springer, 2009.

[58] S. Ivanov, E. Nett and R. Schumann. Fault-tolerant Base Station Planning of Wireless Mesh Networks in Dynamic Industrial Envoronments. In *Proc. IEEE Conf. Emerging Technologies and Factory Automation (ETFA'10)*, Sept. 2010.

[59] R. K. Jain, D. M. Chiu and W. R. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System. Technical report, Digital Equipment Corporation, 1984.

[60] K. Karenos, D. Pendarakis, V. Kalogeraki, H. Yang and Z. Liu. Overlay Routing under Geographically Correlated Failures in Distributed Event-Based Systems. In *Proc. 2010 Int'l Conf. On the Move to Meaningful Internet Systems (OTM'10)*, pages 764–784, Oct. 2010.

[61] A. Kashyap, S. Khuller and M. Shayman. Relay Placement for Fault Tolerance in Wireless Networks in Higher Dimensions. *Computational Geometry: Theory and Applications*, Volume 44, Number 4, pages 206–215, May 2011.

[62] A. Kchiche and F. Kamoun. Access-Points Deployment for Vehicular Networks Based on Group Centrality. In *Proc. 3rd Int'l Conf. New Technologies, Mobility and Security (NTMS'09)*, pages 1–6, Dec. 2009.

[63] A. Kchiche and F. Kamoun. Centrality-based Access-Points Deployment for Vehicular Networks. In *Proc. 17th Int'l Conf. Telecommunications (ICT'10)*, pages 700–706, Apr. 2010.

[64] Y. Kim, T. Schmid, Z. M. Charbiwala, J. Friedman and M. B. Srivastava. NAWMS: Nonintrusive Autonomous Water Monitoring System. In *Proc. 6th ACM Conf. Embedded Networked Sensor Systems (SenSys'08)*, pages 309–322, Nov. 2008.

[65] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley, 2011.

[66] J. Ko, C. Lu, M. Srivastava, J. Stankovic, A. Terzis and M. Welsh. Wireless Sensor Networks for Healthcare. *Proc. of the IEEE*, Volume 98, Number 11, pages 1947–1960, Nov. 2010.

[67] R. Kumar, R. Crepaldi, H. Rowaihy, A. F. Harris III, G. Cao, M. Zorzi and T. F. La Porta. Mitigating Performance Degradation in Congested Sensor Networks. *IEEE Transactions on Mobile Computing*, Volume 7, Number 6, pages 682–697, Jun. 2008.

[68] K. Langendoen and G. Halkes. Energy-Efficient Medium Access Control. In R. Zurawski (editor), *Embedded Systems Handbook*, pages 34.1–34.29. CRC Press, 2005.

[69] S. Lindsey and C. S. Raghavendra. PEGASIS: Power-Efficient Gathering in Sensor Information Systems. In *Proc. 3rd IEEE Int'l Conf. Communications (ICC'01)*, Jun. 2001.

[70] H. Liu, P. Wan and X. Jia. On Optimal Placement of Relay Nodes for Reliable Connectivity in Wireless Sensor Networks. *Combinatorial Optimization*, Volume 11, Number 2, pages 249–260, Mar. 2006.

[71] E. L. Lloyd and G. Xue. Relay Node Placement in Wireless Sensor Networks. *IEEE Transactions on Computers*, Volume 56, Number 1, pages 134–138, Jan. 2007.

[72] D. Luo, H. Zhao, Y. Bi, K. Lin, X. Zhang, Z. Yin and P. Sun. Multiple Sink Nodes' Deployment Based on PMP in WSNs. In *Proc. 2nd Int'l Conf. Pervasive Computing and Applications (ICPCA'07)*, pages 675–678, Jul. 2007.

[73] S. Mahmud, H. Wu and J. Xue. Efficient Energy Balancing Aware Multiple Base Station Deployment for WSNs. In P. J. Marrn and K. Whitehouse (editors), *Proc. 8th European Conf. Wireless Sensor Networks (EWSN'11)*, Volume 6567 LNCS, pages 179–194, Feb. 2011.

[74] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proc. 1st ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA'02)*, pages 88–97, Sept. 2002.

[75] D. Malan, T. Fulford-Jones, M. Welsh and S. Moulton. CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In *Proc. Workshop on Applications of Mobile Embedded Systems (WAMES'04)*, Jun. 2004.

[76] M. Maróti, B. Kusy, G. Simon and Á. Lédeczi. Robust Multi-Hop Time Synchronization in Sensor Networks. In *Proc. Int'l Conf. Wireless Networks (ICWN'04)*, pages 454–460, Jun. 2004.

[77] S. L. Martins, P. M. Pardalos, M. G. C. Resende and C. C. Ribeiro. Greedy Randomized Adaptive Search Procedures for the Steiner Problem in Graphs. In P. M. Pardalos, S. Rajasekaran and J. Rolim (editors), *Randomization Methods in Algorithmic Design*, Volume 43 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 133–145. American Mathematical Society, 1999.

[78] S. L. Martins, M. G. C. Resende and C. C. Ribeiro P. M. Pardalos. A Parallel GRASP for the Steiner Tree Problem in Graphs Using a Hybrid Local Search

Strategy. *Journal of Global Optimization*, Volume 17, Number 1-4, pages 267–283, Sept. 2000.

[79] T. Melodia, M. C. Vuran and D. Pompili. The State of the Art in Cross-Layer Design for Wireless Sensor Networks. In *Proc. EuroNGI Workshops Wireless and Mobility*, pages 78–92, Jul. 2005.

[80] Z. Merhi, M. Elgamel and M. Bayoumi. EB-MAC: An Event Based Medium Access Control for Wireless Sensor Networks. In *Proc. 2009 IEEE Int'l Conf. Pervasive Computing and Communications (PerCom'09)*, pages 1–6, Mar. 2009.

[81] S. Misra, S. D. Hong, G. Xue and J. Tang. Constrained Relay Node Placement in Wireless Sensor Networks to Meet Connectivity and Survivability Requirements. In *Proc. 27th Ann. IEEE Conf. Computer Communications (INFOCOM'08)*, pages 281–285, Apr. 2008.

[82] X. Ning and C. G. Cassandras. Optimal Cluster-Head Deployment in Wireless Sensor Networks with Redundant Link Requirements. In *Proc. 2nd Int'l Conf. Performance Evaluation Methodologies and Tools (ValueTools'07)*, Oct. 2007.

[83] S. H. Oh, C. O. Hong and Y. H. Choi. A Malicious and Malfunctioning Node Detection Scheme for Wireless Sensor Networks. *Wireless Sensor Network*, Volume 4, Number 3, pages 84–90, 2012.

[84] E. M. R. Oliveira, H. S. Ramos and A. A. F. Loureiro. Centrality-based Routing for Wireless Sensor Networks. In *Proc. 3th Int'l Conf. IFIP Wireless Days (WD'10)*, pages 1–5, Oct. 2010.

[85] E. I. Oyman and C. Ersoy. Multiple Sink Network Design Problem in Large Scale Wireless Sensor Networks. In *Proc. IEEE Int'l Conf. Communications (ICC'04)*, pages 3663–3667, Jun. 2004.

[86] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[87] P. H. Pathak and R. Dutta. Impact of Power Control on Relay Load Balancing in Wireless Sensor Networks. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC'10)*, pages 1–6, Apr. 2010.

[88] J. Polastre, J. Hill and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proc. 2nd Int'l Conf. Embedded Networked Sensor Systems (SenSys'04)*, pages 95–107, Nov. 2004.

[89] R. Prasad and H. Wu. Gateway Deployment Optimization in Cellular Wi-Fi Mesh Networks. *Journal of Networks*, Volume 1, Number 3, pages 31–39, Jul. 2006.

[90] J. Pu, Z. Xiong and X. Lu. Fault-Tolerant Deployment with k-connectivity and Partial k-connectivity in Sensor Networks. *Wireless Communications and Mobile Computing*, Volume 9, Number 7, pages 909–919, May 2008.

[91] L. Qiu, R. Chandra, K. Jain and M. Mahdian. Optimizing the Placement of Integration Points in Multi-hop Wireless Networks. In *Proc. 12th IEEE Int'l Conf. Network Protocols (ICNP'04)*, pages 271–282, Oct. 2004.

[92] S. Rajasegarar, C. Leckie and M. Palaniswami. Anomaly Detection in Wireless Sensor Networks. *IEEE Wireless Communications*, Volume 15, Number 4, pages 34–40, 2008.

[93] V. Rajendran, J. J. Garcia-Luna-Aceves and K. Obraczka. Energy-Efficient, Application-Aware Medium Access for Sensor Networks. In *Proc. 2nd IEEE Int'l Conf. Mobile Adhoc and Sensor Systems (MASS'05)*, Nov. 2005.

[94] V. Rajendran, K. Obraczka and J. J. Garcia-Luna-Aceves. Energy-Efficient Collision-Free Medium Access Control for Wireless Sensor Networks. In *Proc. 1st Int'l Conf. Embedded Networked Sensor Systems (SenSys'03)*, pages 181–192, Nov. 2003.

[95] R. Ravi and D. P. Williamson. An Approximation Algorithm for Minimum-Cost Vertex-Connectivity Problems. *Algorithmica*, Volume 18, Number 1, pages 21–43, 1997.

[96] M. G. C. Resende and C. C. Ribeiro. Greedy Randomized Adaptive Search Procedures. In F. Glover and G. Kochenberger (editors), *State of the Art Handbook in Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2002.

[97] I. Rhee, A. Warrier, M. Aia and J. Min. Z-MAC: A Hybrid MAC for Wireless Sensor Networks. In *Proc. 3rd Int'l Conf. Embedded Networked Sensor Systems (SenSys'05)*, pages 90–101, Nov. 2005.

[98] M. Ringwald and K. Römer. BurstMAC - An Efficient MAC Protocol for Correlated Traffic Bursts. In *Proc. 6th Int'l Conf. Networked Sensing Systems (INSS'09)*, pages 1–9, Jun. 2009.

[99] J. Segovia, E. Calle and P. Vila. An Improved Method for Discovering Link Criticality in Transport Networks. In *Proc. 6th Int'l Conf. Broadband Communications, Networks, and Systems (BROADNETS'09)*, pages 1–8, Sept. 2009.

[100] S. Severi and D. Dardari. Performance Limits of Time Synchronization in Wireless Sensor Networks. In *Proc. IEEE Int'l Conf. Communications (ICC'08)*, pages 2124–2128, May 2008.

[101] Y. Shavitt and Y. Singer. Beyond Centrality - Classifying Topological Significance using Backup Efficiency and Alternative Paths. In *Proc. 6th Int'l IFIP-TC6 Conf. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet (NETWORKING'07)*, pages 774–785, May 2007.

[102] Y. Shavitt and Y. Singer. Beyond Centrality - Classifying Topological Significance using Backup Efficiency and Alternative Paths. *New Journal of Physics*, Volume 9, Number 266, Aug. 2007.

[103] I. Slama, B. Jouaber and D. Zeghlache. Energy Efficient Scheme for Large Scale Wireless Sensor Networks with Multiple Sinks. In *Proc. IEEE Wireless Communications and Networking Conf. (WCNC'08)*, pages 2367–2372, Mar. 2008.

[104] IEEE Computer Society. *802.11 IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Jun. 2007. Available at `http://standards.ieee.org/about/get/802/802.11.html` [2 August 2012].

[105] B. Son, Y. S. Her and J. G. Kim. A Design and Implementation of Forest-Fires Surveillance System based on Wireless Sensor Networks for South Korea Mountains. *Int'l Journal of Computer Science and Network Security*, Volume 6, Number 9B, pages 124–130, Sept. 2006.

[106] A. Srinivas and E. Modiano. Minimum Energy Disjoint Path Routing in Wireless Ad-hoc Networks. In *Proc. 9th Ann. Int'l Conf. Mobile Computing and Networking (MobiCom'03)*, pages 122–133, Sept. 2003.

[107] K. Srinivasan and P. Levis. RSSI is Under Appreciated. In *Proc. 3rd Workshop Embedded Networked Sensors (EmNets'06)*, May 2006.

[108] I. Stojmenovic (editor). *Handbook of Sensor Networks - Algorithms and Architectures*. John Wiley & Sons, Inc., 2005.

[109] T. Tabirca, K. N. Brown and C. J. Sreenan. A Dynamic Model for Fire Emergency Evacuation Based on Wireless Sensor Networks. In *Proc. 8th Int'l Symp. Parallel and Distributed Computing (ISPDC'09)*, Jul. 2009.

[110] J. Tang, B. Hao and A. Sen. Relay Node Placement in Large Scale Wireless Sensor Networks. *Computer Communications*, Volume 29, Number 4, pages 490–501, Feb. 2006.

[111] M. Tang. Gateways Placement in Backbone Wireless Mesh Networks. *Int'l Journal Communications, Networks and System Sciences*, Volume 2, Number 1, pages 44–50, Feb. 2009.

[112] J. Tarng, B. Chuang and P. Liu. A Relay Node Deployment Method for Disconnected Wireless Sensor Networks: Applied in Indoor Environments. *Journal of Network and Computer Applications*, Volume 32, Number 3, pages 652–659, May 2009.

[113] D. Torrieri. Algorithms for Finding an Optimal Set of Short Disjoint Paths in a Communication Network. *IEEE Transactions on Communications*, Volume 40, Number 11, pages 1698–1702, Nov. 1992.

[114] T. van Dam and K. Langendoen. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proc. 1st Int'l Conf. Embedded Networked Sensor Systems (SenSys'03)*, pages 171–180, Nov. 2003.

[115] Z. Vincze, R. Vida and A. Vidacs. Deploying Multiple Sinks in Multi-hop Wireless Sensor Networks. In *Proc. IEEE Int'l Conf Pervasive Services (ICPS'07)*, pages 55–63, Jul. 2007.

[116] VINT. *The ns Manual (formerly ns Notes and Documentation)*, Jan. 2009. Available at `http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf` [30 April 2010].

[117] Q. Wang. Packet Traffic: A Good Data Source for Wireless Sensor Network Modeling and Anomaly Detection. *IEEE Network*, Volume 25, Number 3, pages 15–21, 2011.

[118] Y. Wang, C. Hu and Y. Tseng. Efficient Deployment Algorithms for Ensuring Coverage and Connectivity of Wireless Sensor Networks. In *Proc. 1st Int'l Conf. Wireless Internet (WICON'05)*, pages 114–121, Jul. 2005.

[119] S. Wenming, H. Chuanhe, S. Mingkai, C. Yong and C. Zhe. Indoor Localization Scheme in Wireless Sensor Networks Using Spatial Information. In

Proc. Int'l Conf. Wireless Communications, Networking and Mobile Computing (WiCOM'06), pages 1–5, Sept. 2006.

[120] J. L. Wong, R. Jafari and M. Potkonjak. Gateway Placement for Latency and Energy Efficient Data Aggregation. In *Proc. 29th Ann. IEEE Int'l Conf. Local Computer Networks (LCN'04)*, pages 490–497, Nov. 2004.

[121] C. H. Wu, K. C. Lee and Y. C. Chung. A Delaunay Triangulation Based Method for Wireless Sensor Network Deployment. *Computer Communications*, Volume 30, Number 14–15, pages 2744–2752, Oct. 2007.

[122] X. Xu and W. Liang. Placing Optimal Number of Sinks in Sensor Networks for Network Lifetime Maximization. In *Proc. IEEE Int'l Conf. Communications (ICC'11)*, Jun. 2011.

[123] W. Ye, J. Heidemann and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proc. 21st Ann. Joint Conf. IEEE Computer and Communications Societies (INFOCOM'02)*, pages 1567–1576, Jun. 2002.

[124] W. Youssef and M. Younis. Intelligent Gateway Placement for Reduced Data Latency in Wireless Sensor Networks. In *Proc. IEEE Int'l Conf. Communications (ICC'07)*, pages 3805–3810, Jun. 2007.

[125] Y. Zeng, C. J. Sreenan, L. Sitanayah, N. Xiong, J. H. Park and G. Zheng. An Emergency-Adaptive Routing Scheme for Wireless Sensor Networks for Building Fire Hazard Monitoring. *Sensors*, Volume 11, Number 3, pages 2899–2919, Mar. 2011.

[126] W. Zhang, G. Xue and S. Misra. Fault-Tolerant Relay Node Placement in Wireless Sensor Networks: Problems and Algorithms. In *Proc. 26th Ann. IEEE Conf. Computer Communications (INFOCOM'07)*, pages 1649–1657, May 2007.

[127] T. Zheng, S. Radhakrishnan and V. Sarangan. PMAC: An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proc. 19th IEEE*

Int'l Symp. Parallel and Distributed Processing (IPDPS'05), pages 65–72, Apr. 2005.

[128] P. Zhou, B. S. Manoj and R. Rao. A Gateway Placement Algorithm in Wireless Mesh Networks. In *Proc. 3rd Int'l Conf. Wireless Internet (WICON'07)*, Oct. 2007.