Congestion and Medium Access Control in 6LoWPAN WSN

by

Vasilis Michopoulos

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

6th September 2012

Abstract

In computer networks, congestion is a condition in which one or more egressinterfaces are offered more packets than are forwarded at any given instant [1]. In wireless sensor networks, congestion can cause a number of problems including packet loss, lower throughput and poor energy efficiency. These problems can potentially result in a reduced deployment lifetime and under-performing applications. Moreover, idle radio listening is a major source of energy consumption therefore low-power wireless devices must keep their radio transceivers off to maximise their battery life-time. In order to minimise energy consumption and thus maximise the lifetime of wireless sensor networks, the research community has made significant efforts towards power saving medium access control protocols with Radio Duty Cycling. However, careful study of previous work reveals that radio duty cycle schemes are often neglected during the design and evaluation of congestion control algorithms. This thesis argues that the presence *(or lack)* of radio duty cycle can drastically influence the performance of congestion control mechanisms. To investigate if previous findings regarding congestion control are still applicable in IPv6 over low power wireless personal area and duty cycling networks; some of the most commonly used congestion detection algorithms are evaluated through simulations. The research aims to develop duty cycle aware congestion control schemes for IPv6 over low power wireless personal area networks. The proposed schemes must be able to maximise the networks goodput, while minimising packet loss, energy consumption and packet delay. Two congestion control schemes, namely DCCC6 *(Duty Cycle-Aware Congestion Control for 6LoWPAN Networks)* and CADC *(Congestion Aware Duty Cycle MAC)* are proposed to realise this claim.

DCCC6 performs congestion detection based on a dynamic buffer. When congestion occurs, parent nodes will inform the nodes contributing to congestion and rates will be readjusted based on a new rate adaptation scheme aiming for local fairness. The child notification procedure is decided by DCCC6 and will be different when the network is duty cycling. When the network is duty cycling the child notification will be made through unicast frames. On the contrary broadcast frames will be used for congestion notification when the network is not duty cycling. Simulation and test-bed experiments have shown that DCCC6 achieved higher goodput and lower packet loss than previous works. Moreover, simulations show that DCCC6 maintained low energy consumption, with average delay times while it achieved a high degree of fairness.

CADC, uses a new mechanism for duty cycle adaptation that reacts quickly to changing traffic loads and patterns. CADC is the first dynamic duty cycle protocol implemented in Contiki Operating system *(OS)* as well as one of the first schemes designed based on the arbitrary traffic characteristics of IPv6 wireless sensor networks. Furthermore, CADC is designed as a stand alone medium access control scheme and thus it can easily be transfered to any wireless sensor network architecture. Additionally, CADC does not require any time synchronisation algorithms to operate at the nodes and does not use any additional packets for the exchange of information between the nodes *(For example no overhead)*.

In this research, 10000 simulation experiments and 700 test-bed experiments have been conducted for the evaluation of CADC. These experiments demonstrate that CADC can successfully adapt its cycle based on traffic patterns in every traffic scenario. Moreover, CADC consistently achieved the lowest energy consumption, very low packet delay times and packet loss, while its goodput performance was better than other dynamic duty cycle protocols and similar to the highest goodput observed among static duty cycle configurations.

Keywords: Sensor networks; MAC; Congestion control; Duty cycle; Channel check rate; Energy conservation; 6LoWPAN.

Acknowledgements

I am most grateful to my supervisors, Dr Lin Guan and Dr Iain Phillips for their support and guidance throughout the journey of my PhD research. During the three years of this study they provided me with invaluable feedback on my research. It was a joyous experience to work with them.

I would like to thank Dr George Oikonomou for his support through this project and his valuable contributions to Contiki OS for CC2430 devices. He is a friend and personally; I benefited both academically and technically from his experience during the three years of my study.

I would also like to thank Loughborough University in general and the Department of Computer Science in particular for funding my PhD studies, without this support, this work could not have been done.

I would like to extend my gratitude to Computer Science department staff for their help in facilitating my research.

I would like to thank the members of my research group, other research students and staff in my department for making the office a great place to work and develop friendships.

I would like to thank my family members and Gizem Osman for supporting me through the difficult times of my PhD.

This work is dedicated to my beloved parents who have provided me with the best support since I was born.

Publications

Journal Publications

- **V. Michopoulos**, L. Guan, G. Oikonomou, I. Phillips, "CADC: A Congestion Aware Duty Cycle Mechanism for 6LoWPAN Networks", To be submitted by the end of 2012

- **V. Michopoulos**, L. Guan, G. Oikonomou, I. Phillips, "Testbed Evaluation of the CADC Algorithm and the Benefits of Dynamic Duty Cycling in 6LoWPAN Networks", Springer International Journal of Wireless Information Networks, 2012[0]

Conference Publications

- **V. Michopoulos**, L. Guan, G. Oikonomou, I. Phillips, "DCCC6: Duty Cycle-aware congestion control for 6LoWPAN networks," Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on , vol., no., pp.278-283, 19-23 March 2012 doi: 10.1109/PerComW.2012.6197495

- **V. Michopoulos**, L. Guan, G. Oikonomou, and I. Phillips, "A comparative study of congestion control algorithms in IPv6 Wireless Sensor Networks," Distributed Computing in Sensor Systems and Workshops, International Conference on, pp. 1-6, 2011 IEEE International Conference on Distributed Computing in Sensor Systems and Workshops, 2011

- **V. Michopoulos**, L. Guan, and I. Phillips, A New Congestion Control Mechanism for WSNs. In Proceedings of the 2010 10th IEEE international Conference on Computer and information Technology (June 29 - July 01, 2010). CIT. IEEE Computer Society, Washington, DC, pp. 709-714. DOI= http://dx.doi.org/10.1109/CIT.2010.138

Contents

Abstract **4**

Acknowledgements **6**

Publications **7**

1 Introduction **21**

1.1 Introduction 21
1.2 Research Aims and Objectives 22
1.3 Original Contributions 23
1.4 Research Methodology 25
1.5 Thesis Outline 28

2 Background and Related Work **29**

2.1 Introduction 29
2.2 Wireless Sensor Networks 29
2.2.1 IP Against Semi-closed Protocol Stacks 30
2.2.2 IPv6 and WSN 31
2.2.3 Routing Over Low Power and Lossy Networks 31
2.2.4 Duty Cycle 32
2.3 An Introduction to Congestion Control 32
2.3.1 Congestion in WSN 33
2.4 Congestion Control Schemes for WSN 34
2.4.1 Collision Avoidance Schemes 36
2.4.2 Rate Adaptation Schemes 38
2.4.3 Path Adaptation Schemes 45
2.4.4 Dynamic Duty Cycle Adaptation Schemes 48
2.4.5 A Detailed Comparison of Congestion Control Schemes . . . 51
2.5 Summary and Discussions 59

3 Experimental Setup and Empirical Practice **62**

3.1 Introduction 62

3.2 Contiki Operating System 63
3.2.1 Contiki Structure 63
3.2.2 Why Contiki? 67
3.3 COOJA Simulator 68
3.4 Sensinode and the CC2431 System-on-Chip 69
3.4.1 8051 CPU and it's Memory 70
3.5 IEEE 802.15.4 73
3.6 Radio Duty Cycle 75
3.6.1 Radio Duty Cycle in Contiki 78
3.7 6LoWPAN 82
3.8 uIP other IP Stacks for Embedded Systems 84
3.9 RPL: Routing Over Low Power and Lossy Networks 86
3.9.1 RPL What Went Wrong? 88
3.10 Setting Up a Test-bed 89
3.11 WSN Application layer and Data Collection Server 95
3.11.1 WSN Application layer 95
3.11.2 Data Collection Server 99

4 Congestion Detection (CD) in Duty Cycle and 6LoWPANs 100
4.1 Introduction 100
4.2 Congestion Detection Approaches 101
4.3 Simulation Configurations 103
4.4 Simulation Results 104
4.4.1 Performance Analysis of the Evaluated Congestion Detection Mechanisms Without Duty Cycle 105
4.4.2 Performance Analysis of the Evaluated Congestion Detection Mechanisms With Duty Cycle 106
4.4.3 Overall Performance Analysis 109
4.5 Summary and Discussions 110

5 DCCC6: Duty Cycle-Aware Congestion Control (CC) for 6LoWPANs111
5.1 Introduction 111
5.2 DCCC6 Design Considerations 112
5.3 Implementation of DCCC6 113
5.3.1 Congestion Detection 113
5.3.2 Rate Adaptation 117
5.3.2.1 Rate Reduction 117
5.3.2.2 Rate Increase 117
5.4 Simulation Experiments 119

5.4.1 Simulation Configurations . . . 119
5.4.2 Simulation Results . . . 120
5.5 Testbed Experiments . . . 127
5.5.1 Testbed Configurations . . . 127
5.5.2 Testbed Results . . . 128
5.6 Simulation Vs Testbed . . . 131
5.7 Summary and Discussions . . . 135

6 CADC: Congestion Aware Duty Cycle MAC 136
6.1 Introduction . . . 136
6.2 CADC Design Considerations . . . 138
6.3 Implementation of CADC . . . 140
6.3.1 General Characteristics and Rules . . . 140
6.3.2 Measuring Congestion Levels and Cycle Adaptation . . . 141
6.3.3 Duty Cycle and Information Sharing . . . 145
6.3.4 CADC Packet Retransmission Scheme . . . 147
6.4 Evaluation of CADC . . . 149
6.5 Simulation Experiments . . . 152
6.5.1 Simulation Configurations . . . 152
6.5.2 Simulation Results . . . 154
6.5.2.1 Goodput . . . 154
6.5.2.2 Packet Loss . . . 162
6.5.2.3 Packet Delay . . . 163
6.5.2.4 Energy Consumption . . . 169
6.5.2.5 Memory Requirements . . . 175
6.5.2.6 Simulation Result Analysis . . . 175
6.6 Test-bed Experiments . . . 176
6.6.1 Test-bed Configurations . . . 176
6.6.2 Testbed Results . . . 178
6.6.2.1 Goodput . . . 178
6.6.2.2 Packet Loss . . . 181
6.6.2.3 Packet Delay . . . 182
6.6.2.4 Energy Consumption . . . 185
6.6.2.5 Memory Requirements . . . 189
6.6.2.6 Test-bed Analysis . . . 190
6.7 Simulation Vs Testbed . . . 190
6.7.1 Goodput . . . 193
6.7.2 Packet Loss . . . 196
6.7.3 Packet Delay . . . 197

6.7.4 Energy Consumption 199
6.7.4.1 When Network is Idle 199
6.7.4.2 Network is Active 200
6.7.5 Memory Requirements 201
6.7.6 Limitations 202
6.8 Summary and Discussions 204

7 Conclusions and Future Work 206
7.1 Conclusions 206
7.2 Future Work 209

References 226

A Data Collection at the Base Station 226

B Test-bed Tools 229

C Simulation Automations and Result Mining 235

D Sensinode - CC2430 Constrains and Code Optimisations 242

E CADC 802.15.4 Packet Framer Modifications 247

F Pseudocodes 250
F.1 Application Layer Pseudocodes 250
F.2 Data collection server Pseudocodes 252
F.3 DCCC6 Pseudocodes 255
F.3.1 CADC Pseudocodes 259

G Logs of CADC's Validation Through Simulation Experiments 264

List of Figures

1.1 Stages of methodology 27

2.1 Many to one traffic in WSN 35
2.2 Disordered traffic in traditional network 35
2.3 Percentage of the overall congestion research field over the years for the different categories 56

3.1 Structure of Contiki 64
3.2 The Contiki core 64
3.3 Networking in Contiki 65
3.4 MAC and RDC layers in Contiki 66
3.5 Contiki with uIP6 network stack 66
3.6 COOJA can simulate at several levels [2] 68
3.7 Example of various level nodes in COOJA [2] 69
3.8 N601 Nanorouter USB and N740 Nanosensor from Sensinode 70
3.9 CC2430 CODE memory space 72
3.10 CC2430 XDATA memory space 73
3.11 IEEE 802.15.4 protocol stack [3] 74
3.12 an example of a 20% duty cycle 77
3.13 Energy consumption of the individual ContikiMAC operations [4] . 78
3.14 The network radio duty cycle with ContikiMAC, averaged for all nodes ta the network without path loss [4] 79
3.15 The network radio duty cycle with ContikiMAC, averaged for all nodes in a network with path loss [4] 80
3.16 The radio duty cycle in a data collection network with path loss, with X-MAC and ContikiMAC, as a function of the wake up frequency called channel check rate [4] 81
3.17 6LoWPAN adaptation layer 82
3.18 Layout of 6LoWPAN headers 83
3.19 The memory footprint for uIP and other commercialy available IPv6 stacks [5] 85

3.20 The 26 defined by the IEEE 802.15.4 channels and how they overlap with 802.11 channels . . . 92
3.21 IPv6 and 6LoWPAN WSN . . . 94
3.22 Sensor node application flow chart . . . 98

4.1 Successfully received packets without RDC (SicslowMAC) . . . 106
4.2 Packet Lost in the network (NO RDC - SicslowMAC - always on) . 107
4.3 Successfully received packets with RDC (ContikiMAC) . . . 108
4.4 Packet Lost in the network (With RDC - ContikiMAC) . . . 109

5.1 DCCC6 flow chart . . . 115
5.2 Dynamic buffer in DCCC6 . . . 116
5.3 Comparison between traditional AIMD and proposed rate adaptation 119
5.4 Routing tree used in our simulations experiments . . . 122
5.5 Successfully received packets at the sink . . . 124
5.6 Successfully received packets at the sink . . . 125
5.7 Packet delay . . . 125
5.8 Jain's Index fairness over 10 second intervals . . . 126
5.9 Jain's Index fairness over 10 second intervals . . . 126
5.10 Average energy consumption per node *(for the whole duration of the experiment)* . . . 127
5.11 Routing tree used in our test bed experiments . . . 128
5.12 Ratio of successfully received packets over the number of hops . . . 129
5.13 Packet delay (round trip) . . . 130
5.14 Packet delay (round trip) . . . 130
5.15 Goodput: hardware against simulation . . . 131
5.16 Simulation and Test-bed common schenario . . . 132

6.1 CADC Channel check rate adaptation scenarios for different traffic patterns . . . 141
6.2 State transition diagram of CADC's 5 operating states . . . 143
6.3 CADC traffic flow operation scenario . . . 146
6.4 802.15.4 frame header and the 2 bytes of frame control field (FCF) . 147
6.5 CADC packet retransmission scheme . . . 148
6.6 CADC evaluation experiment with normal traffic . . . 150
6.7 CADC evaluation experiments with normal traffic *(part-2)* . . . 151
6.8 Routing tree with coverage *(line topology)* . . . 154
6.9 routing tree *(random 25 node topology)* . . . 155
6.10 Goodput for the different packet transmission intervals with different channel check rates *(line topology)* . . . 158

6.11 Goodput over distance in hops (ContikiMAC&X-MAC best performance. *Channel check rate=64 - line topology*) 158
6.12 Goodput over the different channel check rates (*highest traffic condition with inter packet transmission interval equal to 62.5 ms - random topology)* 159
6.13 Goodput with different packet transmission intervals (ContikiMAC&X-MAC best performance. *Channel check rate=64 - random topology*) 159
6.14 Goodput for the different packet transmission intervals with different channel check rates (*random topology*) 160
6.15 Goodput for the different channel check rates and distances in hops *(the demonstrated performance is during heavy congestion with inter packet transmission interval of 62.5 ms)* 161
6.16 Average packet loss (*line topology*) 162
6.17 Packet loss for the different packet transmission intervals with different channel check rates (*random topology*) 163
6.18 Average delay with different channel check rates *(line topology)* . . 164
6.19 Packet delay over distance in hops (contikiMAC&X-MAC best performance. *Channel check rate=64 - line topology*) 165
6.20 Average packet delay for the different packet transmission intervals with different channel check rates *(random topology)* 165
6.21 Packet delay with different packet transmission intervals (contikiMAC&X-MAC best performance. *Channel check rate=64 - random topology*) 166
6.22 Packet delay for the different channel check rates and distances between 2 - 5 hops *(line topology)* 167
6.23 Packet delay for the different channel check rates and distances between 6 - 9 hops *(line topology)* 168
6.24 Idle network energy consumption/sec for each node 170
6.25 Average energy consumption per successfully received packet *(line topology)* 171
6.26 Average energy consumption/successfully received packet over distance in hops *(for all channel check rates, line topology)* 171
6.27 Average energy consumption/successfully received packet with different packet transmission intervals *(line topology)* 172
6.28 Total energy consumption for the different packet transmission intervals with different channel check rates *(random topology)* 172
6.29 Energy consumption/successfully received packet for the different packet transmission intervals with different channel check rates *(random topology)* 173

6.30 Average energy consumption per successfully received packet with different packet transmission intervals *(random topology)* 173
6.31 Average energy consumption/successfully received packet for the different distances in hops *(line topology)* 174
6.32 Routing tree created by RPL during our test bed experiments . . . 178
6.33 Goodput for different inter packet transmission intervals over channel check rate and distance in hops 179
6.34 goodput for different channel check rates over distance in hops during high traffic configuration *(inter packet transmission interval 62.5 ms)* 180
6.35 Percentage of successfully received packets over different transmission lengths *(Distance is three hops)* 181
6.36 percentage of round trip packet loss during heavy and low traffic over different channel check rate and distance in hops *(categories with 0 loss are not presented for clarity)* 183
6.37 Round trip packet delay during heavy and low traffic over different channel check rate and distance in hops 184
6.38 CADC round trip delay over time in seconds *(2 hops distance - 0.2 sec interval between consequent ping packets)* 185
6.39 Per second energy consumption when network is idle 186
6.40 Average energy consumption for 100 transmitted from the sources packets 187
6.41 Average energy consumption per successfully received packet for 100 transmitted from the sources packets 187
6.42 Network's, per packet average energy consumption over inter packet transmission interval 188
6.43 Network's, per packet average energy consumption over distance in hops 189
6.44 Goodput during simulations *(line topology)* 195
6.45 Goodput during test-bed experiments 195
6.46 Packet delay for different distance in hops with various channel check rate configurations *(simulation)* 198
6.47 Per packet energy consumption over distance in hops *(simulation)* . 201

G.1 Cooja simulator CADC evaluation experiments 268

List of Tables

2.1 Congestion control schemes 52
2.2 Congestion control categories over the years 57
2.3 Congestion detection mechanisms over the years 58
2.4 Congestion detection mechanisms for the different CD categories . . 58
2.5 Congestion sharing approaches over the years 58
2.6 Congestion sharing approaches for the different categories 59

3.1 CC2430 Mapping between physical memory and hardware memory spaces 72

4.1 Number of successfully received packets at the sink 105
4.2 Number of total lost packets (Application wise) 105

5.1 An example of the relation between duty *(expressed as CCR)* and the variables: ϵ, t_{min} for t_{max} = 256t *(clock ticks)* 118
5.2 Packet Loss *(loss due to reception of corrupted packets is not calculated)* 120
5.3 Jain's Index Fairness 121
5.4 Diversity Energy Consumption 121
5.5 Size of the various schemes in bytes 134

6.1 Different simulation configuration permutations 153
6.2 Size of protocols in bytes *(RDC + MAC layer, compiled with MSP430 GCC)* 175
6.3 Different testbed configuration permutations 177
6.4 Protocol code and memory footprints in bytes using SDCC 189
6.5 Energy consumption for the different platmforms 199

List of Algorithms

1 Packet received (application) . . . 250
2 Send packet (application) . . . 251
3 In application's process thread . . . 251
4 Print statistics for a node . . . 252
5 Save nodes in a table . . . 252
6 Search if a node exists in the table . . . 253
7 Receiving packets and collecting the data . . . 254
8 Packet received from upper layers . . . 255
9 Send congestion notification message . . . 255
10 Reduce transmission rates when congestion notification packet is received . . . 256
11 Increase transmission rates after a successful transmission . . . 256
12 Refresh table with interference nodes . . . 256
13 Receiving a packet . . . 257
14 Returning the status of the packet transmission . . . 258
15 Reset congestion parameters . . . 259
16 Update channel check rate and congestion parameters when forwarding state ends . . . 259
17 Reset node's channel check rate back to default when inactive . . . 259
18 Packet is received . . . 260
19 Compares node's channel check rate with the one received from child nodes . . . 260
20 Transmit an unicast packet . . . 261
21 CADC state decision . . . 261
22 Calculate the new channel check rate based on the state in which the node is . . . 262
23 Retrieving the status of a packet's transmission . . . 263

List of Abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACK	Acknowledgment
AIMD	Additive Increase Multiplicative Decrease
BER	Bit error rate
bps	bits per second
BR	Border Router
CA	Congestion Avoidance
CBR	Constant Bit Rate
CC	Congestion Control
CCA	Clear Channel Assessment
CCR	Channel Check Rate
CD	Congestion Detection
CDMA	Code Division Multiple Access
CPI	Clock Per Instruction
CSMA	Carrier Sense Multiple Access
CW	Contention Window
DAO	Destination Advertisement Object
DC	Duty Cycle
DCCC6	Duty Cycle-Aware Congestion Control for 6LoWPAN Networks
DIO	DODAG Information Object

DIS	DODAG Information Solicitation
DODAG	Destination Oriented Directed Acyclic Graph
E2E	End-to-end
ECN	Explicit Congestion Notification
FDMA	Frequency Division Multiple Access
ICN	Implicit Congestion Notification
IEEE	Institute of Electrical and Electronic Engineering
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv6	Internet Protocol version 6
kbps	kilo bits per second
LLN	Low Power and Lossy Network
LLN	Routing for Low Power and Lossy Networks
LPM	Low Power Mode
LQI	Link Quality Indicator
LQL	Link Quality Level
MAC	Medium access Control
Mbps	Mega bits per second
MCU	Micro Controller Unit
MTU	Maximum Transmission Unit
NACK	Negative Acknowledgment
OF	Objective Function
OTA	Over The Air programming
OWD	One Way Delay
PAN	Personal Area Unit

PDU	Protocol Data Unit
PQ	Priority Queuing
QoS	Quality of Service
RDC	Radio Duty Cycle
RPL	Routing Protocol For Low Power and Lossy Networks
RS	Router Solicitation
RSSI	Received Signal Strength Indicator
RTT	Round Trip Time
SD	Standard Deviation
SE	Standard Error
TCP	Transport Control Protocol
TDMA	Time Division Multiple Access
TPC	Transmission Power Control
UDP	User Datagram Protocol
VBR	Variable Bit Rate
WSN	Wireless Sensor Network

Chapter 1

Introduction

1.1 Introduction

A wireless sensor deployment usually consists of multiple nodes monitoring their surrounding environment. Collected data traverse the network towards sink nodes in a multi-hop fashion. Recently, IPv6 over Low power Wireless Personal Area Networks *(6LoWPAN)* and related specifications are becoming increasingly popular in wireless sensor networks *(WSN)* community. Those technologies promise better scalability in terms of number of nodes, due to the extended addressing scheme. Additionally, the integration between 6LoWPAN and the Internet now becomes a question of network layer routing and alleviates the need for application layer gateways [5].

It has also been shown that idle radio listening is a major source of energy consumption [6]. In order to prolong sensor deployment life cycle, research community members have made significant efforts in the development of Radio Duty Cycling *(RDC)* Medium Access Control protocols *(MAC)*. With radio duty cycling, nodes will turn off their transceivers for long periods of time in order to minimise idle listening. Based on a synchronisation protocol, they will turn on their radios almost simultaneously, exchange data and go back to sleep mode. Crucially, the unique characteristics of duty cycle algorithms and 6LoWPAN sensor networks can have a significant impact on the performance of WSN.

The design of many Congestion Control *(CC)* protocols does not take this into account, resulting in different performance under the presence of RDC. Additionally, with some sensor applications such as surveillance, fire detection and object-tracking systems, a sudden event can cause a large amount of packets and therefore network congestion. In such circumstances, MAC protocols with fixed duty cycle will suffer from data loss due to their incapability of adapting to the traffic needs [4, 7, 8, 9]. To reduce packet loss, the network must be configured with a duty cycle aware; congestion control scheme. Another solution to this problem could be the reconfiguration of the network with a higher duty cycle. However, the increased idle radio listening will consume more energy. Hence a duty cycle should be increased during heavy load and decreased when the network is idle.

Recently, significant efforts have been made for the development of dynamic duty cycle MAC protocols [10, 11, 12, 13, 14, 15, 16, 17]. Conceptually, a number of MAC protocols with capabilities of cycle adaptation based on the traffic have been proposed. The majority of previous works, have not carefully considered the unique characteristics of IPv6 and 6LoWPAN WSN in their design.

1.2 Research Aims and Objectives

The aim of this research is to develop congestion aware MAC schemes that can confront congestion, maximise the network's goodput, minimise packet loss, energy consumption and packet delay in IPv6 WSN. Emerging architectures such as IEEE 802.15.4 [18, 19, 20, 21], IPv6 and 6LoWPAN [22, 23] are becoming dominant in wireless sensor networks, therefore the proposed ideas and implementations thereof are based on these architectures. IEEE 802.15.4, is a MAC/PHY protocol for low power and low data rate wireless networks and recently, it is emerging as a popular choice for various monitoring and control applications. IEEE 802.15.4 is aimed at providing cheap, low-power, short- range communications for embedded devices. 802.15.4 MAC can be run in two modes: beaconless and beacon-enabled. This work is focused in the evaluation and development of new MAC layer solutions based on IEEE 802.15.4 beaconless operation. Even though the design of the proposed schemes is based on 802.15.4 *(beacon less operation mode)* and 6LoWPAN architectures, the schemes must be implemented as a generic MAC layer and be transferable to any architecture without major modifications.

The objectives of the research are:

- To investigate the factors which contribute to congestion in WSN at a node level perspective.
- To investigate the various congestion control schemes proposed by the WSN research community. Additionally, a categorisation and a detailed analysis of existing congestion control schemes must be performed.
- To investigate which of the existing congestion control approaches can be transfered and function in WSN operating with Contiki OS [24, 25, 26, 27], 6LoWPAN and 802.15.4 architectures.

- To investigate how the arbitrary traffic patterns of 6LoWPAN networks affect the performance of existing congestion control schemes.
- To investigate the importance of duty cycle in WSN as well as how duty cycle affects the performance of the various congestion control algorithms.
- To develop new MAC layers capable to confront congestion, minimise energy consumption, achieve high goodput, low packet delay and loss without high memory requirements. The developed schemes must also be:
    1. Independent of the routing algorithms and the topology of the network.
    2. Independent of traffic patterns *(uni-directional, bi-directional or both).*
    3. Independent of the network stack and implemented as a stand alone MAC layer.
    4. Independent of the applications running at each sensor node.
    5. Able to automatically detect if the network is configured with a radio duty cycle algorithm and adjust its operation accordingly.

1.3 Original Contributions

This research not only contributes towards the research field of congestion control in WSN but also contains contributions towards the Contiki OS. The contributions of the thesis are outlined as below:

1. In literature numerous studies have suggested various categorisation methods for the existing congestion control schemes. These reports are categorising the existing congestion control schemes based on their general design or the network layer they are implemented *(MAC, transport, cross layer).* Since congestion control schemes incorporate similar congestion detection and avoidance mechanisms, this research study proposes a new categorisation which is based on the congestion avoidance mechanisms adopted by each scheme. A categorisation such as the above mentioned, will allow readers to achieve a broader understanding of congestion control and have a detailed view of the mechanisms used by the existing congestion control protocols.

   In addition to the new categorisation, a detailed analysis of how congestion mechanisms are combined through the different schemes is proposed. Moreover, congestion control schemes and mechanisms are also presented in a time-line form in order to enhance the analysis thereof *(showing the direction wireless sensor community took through the years in the field of congestion control).*

2. This is the first research study, presenting how the various congestion detection mechanisms are affected by the unique characteristics of 6LoWPAN networks and the presence of duty cycle algorithms. It is shown that some of the existing congestion detection mechanisms are not tailored for 6LoWPAN sensor networks and perform poorly when combined with a duty cycle algorithm. This analysis is a significant step towards understanding the performance of various CD mechanisms in a 6LoWPAN environment, with or without duty cycle, and can serve as a basis for future network designers, when facing the decision of which congestion detection mechanism to adopt for their deployment.

3. A proposal for Duty Cycle-Aware Congestion Control for 6LoWPAN Networks *(DCCC6)*. DCCC6 is one of the first congestion control schemes that consider the existence of duty cycle and it is tailored for the characteristics of IPv6 6LoWPAN networks. The proposed scheme dynamically detects if a duty cycle is activated and modifies its functionality accordingly. Furthermore, even though DCCC6 is based in the characteristics of IPv6 and 6LoWPAN networks it is designed as a stand-alone MAC layer and can easily be combined with other protocol stacks. Simulation and testbed results were conducted for the evaluation of DCCC6. The experiments demonstrate that DCCC6 is better than previous works in terms of goodput and packet loss.

4. A proposal for a new Congestion Aware Duty Cycle MAC *(CADC)*. CADC is independent of network topology, routing protocol and the application at each node. Furthermore, CADC is the first dynamic duty cycle protocol designed based on the arbitrary traffic characteristics of 6LoWPAN. Even though CADC is designed for IPv6 and 6LoWPAN networks, it is designed as a stand alone MAC scheme and can easily be transfered to any WSN architecture *(various protocol stacks)*. Over 10000 simulations experiments and 700 test-bed experiments have been conducted for the evaluation of CADC. Through these experiments, it is demonstrated that CADC successfully adapted its cycle based on traffic patterns in every traffic scenario. Furthermore, CADC does not require any time synchronisation algorithms to operate at the nodes and does not use any additional packets for the exchange of information between the nodes *(no overhead)*. Overall, CADC achieved better performance than other works.

5. The schemes proposed in this research study, are the first congestion control works ever implemented in Contiki OS *(as of the time this thesis is*

written). Contiki is an open source operating system currently used by hundreds of companies and research institutions. Moreover, Contiki OS has demonstrated an immense growth during the recent years. Therefore the proposed schemes are an important contribution to the Contiki OS research community and an example for future network mechanism architects. Additionally, the proposed schemes are some of the first works that consider the existence of duty cycle in the network and are tailored for the unique characteristics of IPv6 6LoWPAN networks.

1.4 Research Methodology

Concept development stage is the first stage of this work and it consists of an extensive literature review. As part of the literature review, a categorisation of the existing congestion control schemes has been introduced. Additionally, a detailed analysis of how congestion mechanisms are combined through the different schemes is presented. Moreover, congestion control schemes and mechanisms are presented in the form of a time-line. This enhances the analysis of existing schemes by demonstrating the focus of WSN research community during the last years.

As the second stage, an evaluation of existing mechanisms, architectures and protocols, as well as the design of the proposed congestion control schemes was performed. Additionally, as part of this stage, various WSN architectures, protocols and operating systems were studied and carefully selected for the evaluation of future algorithms. Firstly, the selection of "Standard" and proprietary protocols that the proposed schemes will be evaluated is of great importance. For example, a 6LoWPAN may be the host of a variety of applications which in turn may lead to various traffic paths, packet sizes and destinations. These parameters, should be taken under consideration for the design of a network protocol. This research is mainly focused in IPv6 WSN *(6LoWPAN)* WSN architecture. Many of the IPv6 advantages and the reasons behind this choice of protocol are described in detail in subsection 2.2.1. In comparison to other semi-closed protocol stacks, many of the network manufacturing goliaths are actively supporting 6LoWPAN. Jon Titus, in one of his articles [28] has described that although IPv6 is relatively new, the Internet Protocol has over 30 years of history behind it. Therefore, 6LoWPAN equipment should not encounter any problems with any type of intellectual property *(IP)*. Large companies such as IBM, Sun, Cisco, and Microsoft have heavily invested in IP-based communications. In case IP had hidden patent liabilities, companies such as the above mentioned would have never invested that amount of money. Unfortunately, proprietary or emerging network hardware and software standards might run up against existing patents that engineers have yet to uncover. More articles supporting 6LoWPAN over other architectures include [29, 5, 30] etc. Another important decision for the optimal design of congestion control mechanism is the network layer in which the mechanism will be implemented. In our work, congestion control mechanisms were implemented as a stand alone MACs. Additionally to congestion control, this research studies how radio duty cycle *(RDC)* affects it. Therefore, designing the congestion mechanisms as a MAC layer can justified since radio duty cycle is implemented as part of the MAC layer. Moreover, by implementing the congestion control schemes as a stand alone MAC it is easy to transfer them into any WSN architecture *(various protocol stacks)*. As a final part of this stage, various congestion detection approaches were implemented and evaluated with simulation experiments. This provided valuable information about the behaviour of existing mechanism and how their behaviour gets affected by RDC. The results obtained from the above mentioned experiments had a significant impact on the design of the proposed algorithms.

The third stage includes the validation and evaluation of the proposed algorithms. The proposed schemes were evaluated and validated through both simulation and test-bed experiments under various traffic parameters and network topologies. During the simulation experiments, real hardware nodes operating with the same as the test-bed codes were emulated. This significantly contributes in acquiring realistic results through the simulations. As part of this stage a comparison of the simulation and test-bed results was performed.

Figure 1.1 illustrates the three stages of the methodology in the form of a flow chart.

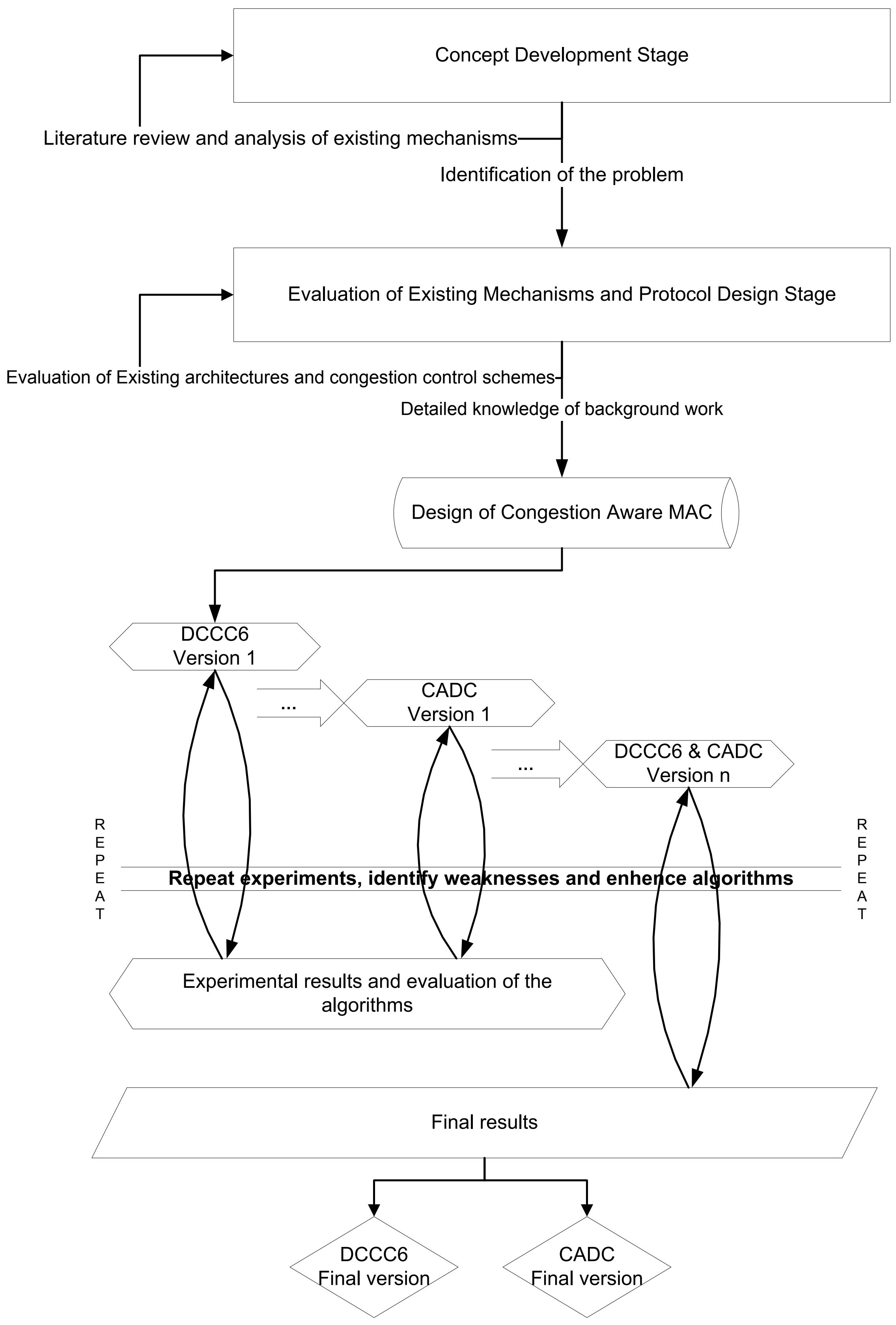
Concept Development Stage
Literature review and analysis of existing mechanisms
Identification of the problem
Evaluation of Existing Mechanisms and Protocol Design Stage
Evaluation of Existing architectures and congestion control schemes
Detailed knowledge of background work
Design of Congestion Aware MAC
DCCC6 Version 1
...
CADC Version 1
...
DCCC6 & CADC Version n
REPEAT
Repeat experiments, identify weaknesses and enhence algorithms
REPEAT
Experimental results and evaluation of the algorithms
Final results
DCCC6 Final version
CADC Final version

Figure 1.1: Stages of methodology

1.5 Thesis Outline

The rest of the thesis is organised as following:

Chapter 2 gives an overview of wireless sensor networks, IP in wireless sensor networks and congestion control. Furthermore, existing congestion control approaches are categorised and summarised. The literature review is then concluded with a detailed analysis of congestion control mechanisms incorporated by the existing schemes and hence motivates the direction of this research.

Chapter 3 presents a detailed overview of the tools and methodology used during this research study. Firstly, the operating system and simulation tools used in this thesis are presented. A detailed description of the hardware used and its architecture follows. Moreover, 802.15.4, radio duty cycle, IPv6 and 6LoWPAN architectures are described in detail. Furthermore, this chapter presents a detailed analysis *(in the form of a guide)* of how to set up a testbed and configuring a WSN for experiments is presented. Finally, the general information concerning the design of our test-bed as well as the software used during the experiments is described in detail.

Chapter 4 presents a detailed analysis of how the various congestion detection mechanisms are affected by the unique characteristics of 6LoWPAN networks and the presence of duty cycle algorithms.

Chapter 5 presents a new Duty Cycle-Aware Congestion Control for 6LoWPAN Networks *(DCCC6)*. DCCC6 is designed as a stand alone MAC for Contiki OS. All of the specific characteristics of 6LoWPAN and duty cycle as well as how these characteristics can affect the design of the scheme are described in this chapter. A detailed analysis of DCCC6's design follows. Finally, simulation and test-bed experiments are presented and evaluated.

Chapter 6 presents a new Congestion Aware Duty Cycle MAC *(CADC)*. CADC is designed as a stand alone MAC/RDC for Contiki OS. A detailed analysis of 6LoWPAN and duty cycling networks as well as what is the constrains in designing a dynamic duty cycle scheme are described in this chapter. Moreover, CADC's design and functionalities are described in detail. Finally, Simulation and test-bed experiments are presented followed by discussions concerning CADC's performance.

Chapter 7 summarises the thesis to show how the aims have been achieved and give suggestions and directions to future work.

Chapter 2

Background and Related Work

2.1 Introduction

2.2 Wireless Sensor Networks

Wireless sensor networks have evolved from the idea that small wireless sensor devices can be used for the collection of information from the physical environment in situations such as wild fire tracking, animal observation, agriculture management, health, military and industrial monitoring. A wireless sensor deployment usually consists of spatially distributed autonomous sensors monitoring their surrounding environment. Collected data traverse the network towards sink nodes in a multi-hop fashion. They are then sent to the end user from the sink node, either directly or through a gateway. Generally, nodes in WSN posses limited processing power and power capabilities while communication is over unreliable and low bandwidth links.

Modern sensor networks are usually using bi-directional communication links which in turn allows additional control to the sensors activity *(for example the administrator may extract data from a specific sensor at any given time).* Bi-directional links with the ability to access nodes individually are usually achieved through sensor architectures such as 6LoWPAN) since each node has its own IP address *(nodes must have a unique address in order to receive data from sources outside the sensor network).* The above statement can also justify why IPv6 and related specifications are becoming increasingly popular in the WSN community. A thorough analysis of how these architectures can affect congestion control mechanisms designed for non IPV6 WSN is presented in subsection 2.2.2. The structure of a modern WSN can be seen in Figure 2.1

During the last years, “Internet of things” has attracted lots of attention from the research community. The Internet of Things predicts a world in which each "thing" is connected to the Internet. One approach for the connection of every "thing" with the Internet is to integrate a wireless sensor in every "thing". IPv6 over 6LoWPAN standard proposes a solution for the Internet Protocol on smart objects *(smart object is an item equipped with a form of sensor actuator, a tiny microprocessor, a communication device and a power source [5]).* This in turn allows for the connection of low-power sensor devices with the Internet [31, 32, 33, 34, 35, 36]. Therefore, IPv6 over 6LoWPAN is a step towards the connection of every device with the Internet.

2.2.1 IP Against Semi-closed Protocol Stacks

A plethora of private or semi-closed protocol stacks have been proposed over the past decade with a purpose of improving the efficiency of networks. These semi-closed protocol stacks usually consists of collapsing layers while there isn't any clear demarcation between the various functions handled by the network protocols. On the contrary, these architectures are proven to be very rigid due to their low link layer dependency and the dependency between the various network functions. Moreover, semi-closed architectures require external mechanisms or add ons usually described as gateways in order to route packets between the Internet and the sensor network [5].

Contradictory to the semi-closed protocol stacks, an IP architecture provides:

- *Scalability:* due to the extended addressing scheme of IP, sensor networks can now accomondate thousands or even millions of nodes.

- *Evolvability:* the IP architecture has proven to be evolvable due to the design of applications, protocols and mechanisms for this architecture. Protocols from different network layers can evolve independently of protocols or mechanisms from other network layers.

- *Diversity of applications:* a WSN technology, tailored for one specific application may not work for other applications. Technologies designed for IP can work with every application designed for an IP network.

- *Interoperability:* the majority of WSN require interoperability between the sensor devices and the existing network infrastructures. With IPv6, the integration between a 6LoWPAN and the Internet now becomes a question of network layer routing and alleviates the need for an application layer gateway.

- *Standardisation:* the mechanisms and protocols that define the operation of a WSN is recommended to be standardised thorough open standards and well established standardisation practices. IP architectures provide open standards and thus lower the entry barrier for the manufacturers and therefore allow them to freely choose between different vendors.

2.2.2 IPv6 and WSN

IPv4 has been deployed at an unimaginable scale and it is currently used by more than a billion devices. In the past few years, it was clear that a new version of IP would be needed due to the exponential growth of IP connected devices. IPv6 is an IP version that intends to succeed IPv4. IPv6 uses 128 bit addresses therefore it has an address space of $3,4 \times 10^{38}$ which is significantly larger to the IPv4's $4,294,967,296$ address space. Even though the migration to IPv6 is inevitable and has already started, the adoption of IPv6 has mainly been delayed due to the migration costs. The need to connect thousands of sensors render IPv6 the protocol of choice for WSN. From an architectural point of view IPv6 is build on the fundamental principles of IPv4 while it offers a significantly larger address space along with other very useful features for WSN such as stateless configuration. The stateless configuration process allows a node to generate its link-local, site local and global addresses by using a combination of local information and information advertised by routers. Due to the usually large size of WSN, stateless auto configuration is well suited to this type of network.

The distinct characteristics of architectures such as IPv6 over 6LoWPAN, can dramatically affect the performance the behavior of standard congestion control mechanisms especially when these mechanisms are designed for semi-closed protocol architectures. A detailed analysis of CC mechanisms and simulations of how these mechanisms perform in a IPv6 sensor network will be presented in chapter 4

2.2.3 Routing Over Low Power and Lossy Networks

In WSN, the communication links are lossy and occasionally unreliable. Packet drops on lossy links are extremely frequent and links may become totally unreliable for periods of time due to interference. The above observations have strong consequences to the design of routing protocols and therefore, the routing protocol should not overreact to failures and attempt to stabilise under unstable conditions. In 2008, the Internet Engineering Task Force *(IETF)* formed a new group called ROLL *(Routing Over Low-Power and Lossy networks)* in order to produce a set of routing requirements and determine whether or not existing IETF routing protocols can satisfy these requirements. The working group observed that none of the existing routing protocols would satisfy the fairly unique set of routing requirements for Low power and Lossy Networks *(LLN)*. This led to the design of the new emerging standard for routing in LLN named Routing for Low Power and Lossy Networks *(RPL)*. RPL will be described in detail in section 3.9

2.2.4 Duty Cycle

Recent studies has shown that idle radio listening is a major source of energy consumption [6]. In order to prolong sensor deployment life cycle, research community members have made significant efforts in the development of RDC MAC. With RDC, nodes will turn off their transceivers for long periods of time in order to minimise idle listening. Based on a synchronisation protocol, they will turn on their radios almost simultaneously, exchange data and go back to sleep mode. The unique characteristics of RDC protocols and how such protocols can affect CC mechanisms is described in detail in subsection 2.2.4, while simulations and a technical review of the problem can be found in chapter 4.

The unique characteristics of duty cycle algorithms have a significant impact on the performance of WSN and more specifically in the various CC mechanisms. A detailed analysis and simulations of how CC mechanisms are affected by RDC algorithms will be presented in chapter 4

2.3 An Introduction to Congestion Control

This chapter presents a comprehensive literature review on congestion control in WSN. The congestion control schemes reviewed are categorised and summarised in Table 2.1 *(on page 52)*. Understanding congestion characteristics and controlling the traffic have become critical with the enormous growth of WSN in recent years.

Numerous methods and different approaches for CC have been suggested in literature. In general, there are three main approaches for CC in WSN and thus a primary categorisation can be made:

1. mechanisms that reduce the frequency of radio collisions.

2. m Congestion mitigation congestion prevention, in which nodes can take multiple actions in order to mitigate or even prevent congestion from occurring.

CC schemes can be implemented in different network layers, or be designed as cross layer implementations; they can also be tailored in the needs of specific applications and protocols or they can be generic. Taking this into consideration, it is easy to conclude that schemes of the same category may achieve the same result *(for example congestion prevention etc.)*, but the procedures and the algorithms employed can be significantly, or even entirely different. When schemes of the same category significantly differ in their design and operation procedures, there can be a great confusion. In the past, numerous studies have suggested alternative categorisation methods with the intention to cover this research field without confusion of the schemes. In this work a new categorisation, based on the operation mechanisms adopted by each scheme is suggested. The categorisation proposed in this work divides the CC schemes in four categories:

1. Collision avoidance schemes.
2. Rate adaptation schemes.
3. Path adaptation schemes.
4. Dynamic Duty Cycle Adaptation Schemes.

The remainder of the chapter is organised as follows:

Subsection 2.3.1 describes congestion control, and how congestion control affect WSN; section 2.4 *(on page* 34 *)* gives an overview of existing CC schemes; A comparison of existing CC schemes is made in subsection 2.4.5 *(on page* 51 *)*; lastly, the literature survey is summarised and discussed in section 2.5 *(on page* 59 *)*.

2.3.1 Congestion in WSN

Network congestion occurs when offered traffic load exceeds available capacity to the degree that quality of service deteriorates. In WSN, congestion can cause a plethora of malfunctions such as packet loss, increased delays, lower throughput and energy inefficiency, potentially resulting in reduced deployment lifetime and under-performing applications. It is evident that congestion in wireless sensor networks mainly leads to two main events: i) buffer overflows and ii) radio collisions *(usually described as channel Contention)*. Collisions can be prevented with Medium Access control approaches, such as with Carrier Sense (CSMA), Time Division (TDMA) and Code Division (CDMA). Reducing the frequency of radio collisions does not necessarily mean that the problem of congestion has been fully resolved. Congestion can still occur, since the local fairness achieved by CSMA protocols frequently contributes to potential buffer overflows.

In addition, several characteristics of WSN contribute significantly to congestion. These characteristics are:

1. Traffic patterns. In traditional networks, traffic is disordered. Figure 2.2 represents the topology as well as the traffic patterns in a traditional wired

network. Data flows can cross with each other since each traffic flow can have a different destination. In comparison, WSN traffic is centripetal *(Tree topology with the root node as the center).* In Figure 2.1 both the topology and possible traffic patterns of a WSN are represented. In many-to-one topologies, all the traffic is forwarded towards the root node *(usually in WSN the sink node is also configured as the root but this is not a necessity).* Considering the difference between the traffic patterns of traditional and sensor networks, it is easy to conclude that the bottleneck of congestion will occurs differently in each case. In a WSN tree topology, nodes closer to the root are more prone to congestion.

2. Sensor node specifications. WSN nodes are hardware constrained with limited amount of memory, low-bandwidth radio and slow microprocessors. Unlike nodes in traditional networks, memory restrictions render complicated algorithms unusable in WSN. Moreover, the more the complexity of the algorithm in a sensor node increases the higher the energy consumption *(due to higher CPU activities)* will be. Algorithms tailored in the needs of WSN should be efficient and simple with minimum processing requirements.

3. Low Power and lossy links (LowPAN's). In a LowPAN, wireless links are unreliable with low-bandwidth *(up to 250 kbps in 802.15.4)* therefore network failures may occur. Furthermore, in wireless links, a transmission is overheard by all the nodes in range, therefore the higher the network density the higher the possibilities of collision. Continuous transmission failures such as: collisions, non acknowledged packets etcetera can result in path changes, routing loops and even congestive collapse of the whole network.

A CC scheme tailored for WSN should be simple enough so it can be applicable in hardware constrained devices, with minimum energy consumption functionalities, aware of the afore mentioned characteristics of WSN and capable to confront any form of congestion.

2.4 Congestion Control Schemes for WSN

In this section, the proposed categorisation of CC schemes is going to be presented in chronological order. The categorisation of the protocols will be made based on the CA mechanisms each CC scheme has embodied. The categories proposed in this thesis are:

- Collision avoidance schemes. Generally, schemes belonging to this category attempt to minimize *(CSMA)* or eliminate *(TDMA, CDMA, FDMA etc.)*

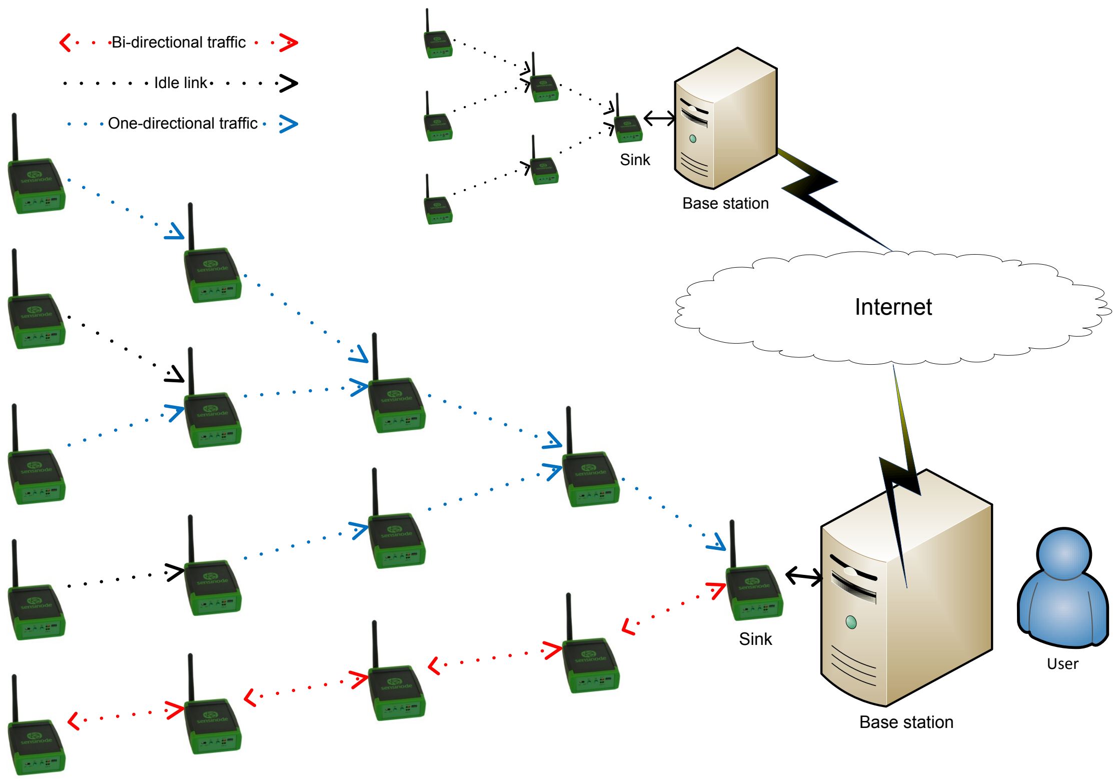

Figure 2.1: Many to one traffic in WSN

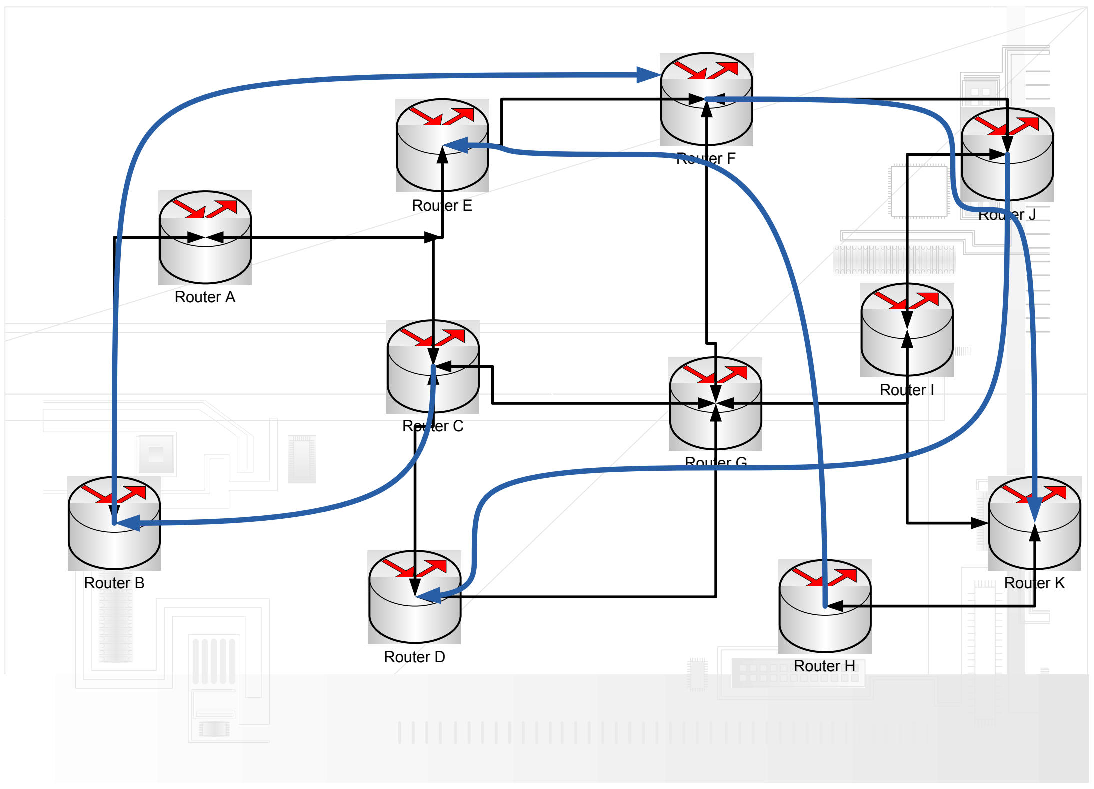

Figure 2.2: Disordered traffic in traditional network

radio collisions. Schemes in this category can incorporate congestion confrontation approaches such as radio collisions prevention.

- Rate adaptation schemes. Schemes in this category adjust their transmission rates based on algorithms such as Additive Increase Multiplicative Decrease *(AIMD)*, in order to control congestion. Usually, rate adaptation schemes have embodied techniques such as Explicit Congestion Notification *(ECN)* and Implicit Congestion Notification *(ICN)* for the exchange of congestion information between nodes. Schemes in this category usually incorporate Congestion mitigation and Congestion avoidance approaches for congestion confrontation.

- Path adaptation schemes. In this category, nodes maintain in their routing tables alternative paths to the destination. Upon congestion, nodes will forward the packets through an alternative path. Schemes in this category can incorporate Congestion mitigation and Congestion avoidance approaches for congestion confrontation.

- Dynamic Duty cycle adaptation schemes. Schemes in this category, adjust their duty cycles based on the traffic load *(when a node increases the duration of radio ON time, the number of packets that can be received or transmitted increases as well)*. With this approach for CC, a WSN can maintain minimum energy consumption with minimum packet losses. Schemes in this category can incorporate congestion confrontation approaches such as Congestion mitigation and Congestion avoidance mechanisms.

2.4.1 Collision Avoidance Schemes

Stathopoulos et al. proposed an application based collision avoidance for wireless sensor networks [37]. Three different mechanisms have been proposed in this work. The first one *uniformed TCP-like collision avoidance* suggests an additive increase and multiplicative decrease *AIMD* based on NACK. For each successful packet transmission the application will additively increase the transmission rate. When a NACK packet is received the rate is decreased. In the second *informed TCP-like collision avoidance*, a similar AIMD scheme is suggested. On the contrary, in this mechanism packet failure information such as collision or link loss are added in the NACK packet. The source will decrease its packet transmission rate only if the packet loss was caused by a collision. Finally a phase-offset collision avoidance mechanism is proposed. This mechanism incorporates a time offset indicating the largest silent period in the NACK packet. Every time the source receives a NACK, a non activity timer is set based on the time offset.

In [38], authors have proposed the Simple, the Adaptive and the Range Adaptive Backoff Protocol *SRBP, ARBP and RARBP* schemes for efficient collision avoidance. All three protocols operate in a similar manner and calculate a back off time for packet retransmission when a collision occurs. In SRBP the back off period is selected uniformly randomly from a continuous space of numbers. This space of numbers is pre-configured. The ARBP protocol is based on the assumption that parameters such as network density and packet transmission rate are known in advance and produces a space of numbers based on these parameters. RARBP protocol adjusts the back off time based on the distance between the sender and the receiver of the message.

A hybrid collision avoidance method is proposed in [39]. In this study each node operates in two alternative modes, sender initiated *(SI)* and receiver initiated *(RI).* SI is the default mode, and uses a four-way RTS-CTS, data, ACK handshake. RI is the newly introduced mode in this hybrid mechanism, which operates with a three-way collision avoidance handshake: i) request for request to send *(RRTS)*, ii) multiple access with collision avoidance by introducing *(MCA-IB)* and iii) collision free receiver initiated multiple access *(RIMA).* A node switches to RI mode only when it does not perform well in SI mode mode. In order to perform receiver initiated handshake, both sender and receiver need to enter RI mode. By adaptively sharing the burden of collision avoidance hand shake between the nodes, better fairness and higher throughput is achieved.

In [40], the authors study carrier sense performance and demonstrate that it can significantly improve the performance in heavy traffic conditions. However, carrier sense has some limitations, originating from the fact that the sender relies on local information to predict the packet reception probability. This can result in lack of information related to parent nodes, which in turn can cause collisions and thus low channel utilisation.

In idle sense [41], each node observes the number of idle slots between two transmission attempts and compares their theoretical estimate. It then adjusts the contention window via an Additive Increase and Multiplicative Decrease *(AIMD)* algorithm. In reality, idle sense is a modification of CSMA/CA; after contention, nodes dynamically converge (in a fully distributed manner) to similar values of their contention window instead of relying on exponential back off. In order to achieve this, a relation between the current state of the network and controlling the contention window is established. Idle sense adjusts the contention window when a collision occurs rather than detecting the collision to control the congestion. Transmission opportunities are allocated to the nodes based on the number of idle slots. This method results in higher throughput and short term fairness.

While traditional CSMA based protocols resolve collisions by backing off in time, Power back off *(CSMA/PB), CSMAPB* enhances CSMA by adding a transmission power control component called power back off (back off in space component). The authors argue that backing off in space is more efficient than backing off in time, demonstrating that by reducing transmission range by 50% results in a four fold decrease in contention. Low transmission range leads to a low contention path and thus higher network throughput. Using this method may lead to extra routing message exchanges in order to adjust the routes to the sink, leading to slightly increased network overhead.

Enhanced CSMA [42], attempts to improve the performance of CSMA by lowering the cost of channel state, by adding a learning approach in order to predict the probability of successful reception. In E-CSMA nodes keep state information about all their neighbours. This is acquired by recording the successful reception probability for each neighbour. Before transmitting a packet, a node uses current channel state and the reserved information of the intended receiver as references.

A contention access method for collision avoidance is suggested in [43]. This work assumes the presence of a DC MAC protocol. In case of collision, all contending sensors reduce their contending probability to half. Based on this, only half of the contending sensors will wake up during the next transmission. This algorithm ensures 50% lower probability for collision each time it is trigered.

In [44], authors have proposed a grant-to-sent approach in order to avoid collisions. Grant-to-send mechanism is implemented as an addition to the traditional CSMA/CA MAC. When a node sends a packet, it also informs other neighbour nodes to remain inactive so they will not collide with the recipient's future transmission. Nodes are also sharing some estimated information about the recipients future actions and thus a higher degree of collision avoidance is achieved.

IBPS: a fault tolerant wireless sensor MAC protocol for efficient collision avoidance is proposed at [45]. When a collision occur, IBPS nodes calculate the back off packet retransmission time and propagate it to all neighbour nodes. Nodes are then calculating future back off periods based on the information received by their neighbours. This mechanism eliminates the probability of collision but adds significant packet overhead.

2.4.2 Rate Adaptation Schemes

Congestion detection *(CD)* and avoidance *(CODA), CODA* is an energy efficient, congestion control scheme for WSN. In CODA, two basic approaches are used in order to confront congestion: Open-loop hop-by-hop backpressure and Closed-loop multi-source regulation. With the former, a backpressure message is generated and transmitted to all one-hop downstream nodes through a broadcast when congestion is detected. When a node receives the backpressure message it decides based on its local network conditions if the message will be further propagated. In Closed-loop multi-source regulation when a source event rate is less than the maximum theoretical throughput of the channel the source regulates itself. On the other hand if the source rate is higher than the maximum theoretical throughput of the channel, closed-loop congestion control is triggered. In that case a source requires constant, slow time-scale feedback (*e.g. ACK*) from the sink in order to maintain its rate. As long as there is no failure to receive ACK, sources maintain their rates; otherwise rate reduction is forced. The mechanisms proposed in this protocol can successfully reduce congestion but they don't eliminate it.

Event to Sink Reliable Transport *(ESRT)* is a transport solution developed to achieve reliable event detection with minimum energy expenditure and congestion resolution functionality [46]. With ESRT, a congestion notification *(CN)* bit is set to the packets header when the buffer is likely to overflow during the next period. The sink based in the reliability measurement, the congestion notification bits and the previous reporting rate will compute the next new reporting rate. In this protocol there is no congestion control mechanism at the intermediate nodes; the sink is responsible upon all rate adjustments in the network. The weakness of this protocol is the need to use a powerful sink node in order to broadcast the rate updates even to the most remote nodes in the network. Furthermore fairness is not considered in this protocol since in case of congestion, rate reduction is applied to all network nodes even when their traffic path is not congested.

In [47] a mitigating congestion control protocol is recommended. This protocol uses three congestion mitigation mechanisms: i) hop-by-hop flow control, ii) source-rate limiting and iii) prioritised medium access layer that allows congestion to drain quickly at local nodes. In rate limiting mechanisms, nodes must continuously monitor their parents actions in order to generate tokens. This passive continuous listening consumes more energy and network resources. In addition, this protocol requires a tree routing structure to work correctly.

In [48] a congestion control and fairness *CCF* for many-to-one routing scheme has been proposed. This congestion control proposal assumes a tree routing structure from the data sources to a sink. This algorithm exists in the transport layer of the traditional network stack model and is designed to work with any MAC. CCF considers two types of congestion and proposes two methods to eliminate them. In the first proposed method, a small jitter is added to the data-link layer in order to achieve small amounts of phase shifting. By implementing a phase shifting technique at the nodes, the probability of collision during simultaneous transmissions from the nodes is reduced. In the second method, the queue occupancy is monitored. When the queue occupancy increases beyond a threshold, the congested node will inform its child nodes about the congestion and a rate reduction mechanism will be triggered at the child nodes. The rate reduction algorithm in this work is suggesting rate allocation based on the number of child nodes: $r_{child} = r_{parent}/n$ where r is the packet transmission rate and n is the number of child nodes.

A reliable bursty convergecast *(RBC)* is proposed in [49]. In RBC, authors have proposed a window-less block acknowledgment as well as a distributed contention control scheme. The former scheme, improves channel utilisation and guarantees continuous packet forwarding irrespective of the underlying link reliability by implementing linked virtual queues. In the later scheme, packet incurred delay is reduced by scheduling the packet retransmissions based on the priority of the virtual queue they have been assigned to.

STCP [50] is a scalable and reliable transport layer protocol for sensor networks. The majority of its functionalities are implemented at the sink. STCP, also supports networks with multiple functionalities such as controlled variable reliability and congestion detection and avoidance. Each node needs to establish an association with the sink; this is achieved with the use of session initiation packets *(session initiation packets contain information such as number of flows, type of date flow, transmission rate and required reliability).* In order to detect congestion, STCP adopts the method of explicit congestion notification. Every sensor node maintains two thresholds in its buffer*(Tlow, Thigh).* When the buffer reaches *Tlow*, it sets the congestion notification bit in packets with a certain probability *(the value of this probability is determined by an approach similar to that employed in RED).* When the buffer reaches *Thigh*, the node set the congestion notification bit to every packet it forwards. On receiving of this packet the sink informs the source of the congested path by setting the congestion bit in the acknowledgment packet. On receiving of the congestion notification packet the sink will slow down the transmission rate.

SenTCP has been proposed in [51]. SenTCP is an open-loop hop-by-hop congestion control protocol and it has two special features: *a)* Congestion degree is estimated by calculating the ratio of the packet service time over the packet inter-arrival time. This ratio is also used in order to differentiate the occurrence of packet loss in the sensor network. *b)* When a node becomes congested, it generates a feedback signal that contains the congestion degree and the queue occupancy. This signal is transmitted to the downstream nodes in a hop-by-hop fashion. When a node receives the feedback signal, based on the received congestion degree and queue occupancy a node will adjust its transmission rate *(in transport layer).*

Chen et al [52] propose an aggregate fairness model which implements end-to-end fairness with a localized algorithm. In order to avoid congestion each node piggybacks its current buffer state in the frame header. When a child node overhears a message it caches the buffer state of it parent node. Child nodes forward packets to parent nodes only if the buffer is not full. Moreover, an aggregate fairness algorithm is used for rate reduction. When a node receives more packets than it can forward, the sensor will calculate and allocate the data rates of child nodes by a weighted fairness function. This means the actual rate from an upstream link should be proportional to the link's aggregate flow weight.

The authors of [53] propose a low-overhead congestion sharing mechanism called Interference-aware Fair Rate Control *(IFRC)*. In IFRC each node adaptively converges to a fair and efficient rate for the flows; with the use of a distributed rate adaptation technique. This is achieved by accurately sharing congestion information with potential interferes (*two nodes are potential interferes when the flow originating from one node uses a link that interferes with the link between the other node and it's parent*). IFRC consists of three inter-related components; one that measures the level of congestion at a node, another that shares congestion information with potential interferes, and a third that adapts rate using an AIMD control law. In order to measure the level of congestion at a node, a single queue for all the flows passing through that node is maintained. When the queue size exceeds an upper threshold (*the upper threshold is dynamically adjusted according to average queue size exchanged between nodes*), the node is said to be congested, and the node reduces its rate according to an AIMD rate adaptation scheme. In order to successfully achieve sharing the average queue size between nodes, the information about the current transmission rate and the average queue size in each node is attached in the header of each outgoing packet.

In [54] a congestion avoidance protocol based on lightweight buffer management is recommended. This protocol, implements a $1/k_{buffer}$ buffer algorithm in order to solve the hidden terminal problem. Sensors are advertising their remaining buffer continuously. A node will transmit a packet only if it overhears that its parent node has enough buffer space for the reception of the packet. Although there is no packet loss; with the use of this algorithm the utilisation of the buffer is very low.

PCCP priority based congestion control for WSN is proposed in [55]. PCCP offers a weighted fairness by offering different degrees of priority to each sensor node. This weighted fairness function allocates more bandwidth to nodes with higher priority index. PCCP further defines the priority index for both self generated and transit traffic. Furthermore, the congestion degree is calculated as the ratio of packet inter-arrival over service time and then imposes hob-by-hop congestion control based on the measured congestion degree and the priority index. PCCP uses ICN by piggybacking the congestion information on the header of the packets. Finally, in PCCP the application can overwrite the priority index at the sensor nodes.

In [56] an adaptive flow and back-off interval that work's in unison with energy efficient, distributed power control (*DPC*) are proposed. The onset of congestion is detected from buffer occupancies at the nodes as well as the predicted transmitting power. Then the rate selection algorithm is executed at the receivers to determine the appropriate rate. Moreover, weights can be assigned to assign different importance between the flows. In addition, in this scheme the distributed power control (*DPC*) and rate information is exchanged between the nodes. The adaptive rate scheme in this protocol is implemented at each node and acts as a back-pressure signal to minimise the effect of congestion on a hop-by-hop basis. First, the outgoing traffic flow is estimated. Consequently the congestion is alleviated by designing a suitable back off intervals for each node based on channel state and current traffic.

RCRT:[57] Rate controlled reliable transport provides reliable, sequenced delivery of flows from the source to the sink. Furthermore RCRT ensures that for a given application, the available network capacity will be allocated according to capacity allocation policy. In RCRT End - to - End reliability is achieved with the use of negative acknowledgments. The sink decides if the network is congested if the time to repair the loss is significantly higher than the round trip time. Once network congestion is determined, the rate adaptation component *(This component is located at the sink)* estimates the total sustainable traffic in the network. Then the rate allocation component *(This component is located at each node, and gets activated upon the reception of a negative acknowledgment)* decreases the flow rates of the sources in order to achieve the rate send by the rate adaptation component. When there is no network congestion rate adaptation component *(in the sources)* additively increase the transmission rates.

HRCCP: hop-by-hop based reliable congestion control has been proposed in [58]. HRCCP uses a pair of end-to-end and hop-by-hop sequence numbers in order to speed up the end-to-end delay. Hop-by-hop sequence numbers are used for the packet retransmissions *(at the MAC layer)* while end-to end sequence is used for the packet reassembly at the sink node. Furthermore, this work has introduced DSbACK feedback messages for CC. A DSbACK will be piggybacked from the parent to the child node in three cases: *1)* an error packet is received, *2)* a timer is out, *3)* the degree of buffer occupancy exceeds the predefined threshold. When a node receives a DSbACK message, it will exponentially decrease its transfer rate.

ABPS, a simple active congestion control protocol is proposed in [59]. This work is designed for many-to-one traffic patterns and needs a tree routing algorithm to operate. In ABPS, when the queue occupancy exceed a predefined threshold, a congestion alleviation mechanism is triggered. When congestion is detected, the congested node will divide its transmission rate with the size of the tree *(at the congested node)* and piggyback this information to its child nodes. When a node receives a congestion message, it will calculate its tree size and set its output transmission rate to the minimum rate based on its newly calculated tree.

In [60] a hybrid congestion control protocol *(HCCP)* is suggested. Each node calculates its congestion degree *(the levels of congestion at the node)* by checking if its packet buffer is likely to overflow during the next period if every packet transmission fail. Nodes periodically exchange their congestion degree which is used for the calculation of the next period's transmission rate. The number of upstream neighbours is also taken into account in this calculation. The periodical exchange of packets consumes more energy and creates additional overhead to the network, especially when the frequency is very high (HCCP's performance improves as frequency increases).

Karenos et al. proposed *COMUT*, a framework that supports multiple classes of traffic for WSN [61]. In COMUT, nodes are organised into classes. The design of COMUT consists of three mechanisms: *1)* cluster formation, *2)* traffic intensity estimation, *3)* rate regulation. In the first mechanism, sensor nodes are organised into clusters and elect a cluster head called sentinel is elected. A zone routing protocol *(ZRP)* is employed to assist in the cluster formation. The second mechanism calculates the traffic intensity within and across multiple clusters and based on that the congestion levels of each cluster are estimated. Once congestion levels are estimated, the third mechanism adjusts the source rates. This is achieved by a communication between the sentinel nodes and the source. COMUT also provides a differentiation between the flows. Low importance flows reduce their rates to the minimum if packets with higher importance exist in the congested path.

In [62], *CTCP:* a collaborative transport control protocol for WSN is proposed. CTCP guarantees reliable packet transmissions between the source and the sink and implements two reliability profiles for energy saving. In order to confront congestion, a CTCP node will broadcast a *stop* message when the queue occupancy exceed a predefined threshold. When a node receives a *stop* message, it will immediately stop transmitting packets to the congested node until the reception of a *start* message.

A priority based congestion control protocol for WSN *(QCCP-PS)* is proposed at [63]. QCCP-PS consists of three parts: *1)* congestion detection unit *(CDU)*, *2)* rate adjustment unit *(RAU)*, *3)* congestion notification unit *(CNU)*. The CDU uses the queue length as the congestion indicator and produces a congestion index which is a number between 0 and 1. Based on the current congestion index and source traffic priority, the RAU calculates the new rate of each child traffic sources as well as its local traffic source. The new rate is then sent to CNU which is responsible for notifying all the child nodes about the new rate. CNU achieves that with ICN, the new rate for each child node is added to the sending data of each sensor node.

In *FACC* [64] a rate-based fairness-aware congestion control is proposed. In this protocol the intermediate relaying sensor nodes are categorised into near-sink and near-source nodes. Near-source nodes maintain a per-flow state and allocate an approximately fair rate to each passing flow by comparing the incoming rate of each flow and the fair bandwidth share. On the other hand, near-sink nodes do not need to maintain a per-flow state and use lightweight probabilistic dropping algorithm based on queue occupancy and hit frequency. This categorisation allows an appropriate rate to be assigned to the near-source nodes, while energy saving and congestion avoidance is secured to the near-sink nodes by a simple algorithm.

Wang et al. proposed upstream hop-by-hop congestion control *(UHCC)* in [65]. UHCC is a cross layer design and operate in two phases: CD and rate adjustment. Based on the queue length and the traffic rate at MAC layer, UHCC calculates a congestion index. All upstream traffic rates are then adjusted according to the calculated congestion index.

LACAS an adaptive learning solution for congestion avoidance is proposed in [66]. The target of this work is to control the data rate of intermediate nodes in order to avoid congestion. To achieve this, a code capable of taking intelligent actions *(called automata)* is implemented at each node. Automata are adjusting the data rates of the intermediate nodes based on the probability of how many packets are likely to be dropped if the data rate in the node remains the same. Automata learn from past behaviors and allocates more accurate data rates in the future.

Extended DCCP is proposed in [67]. In this work, the authors have extended the existing datagram congestion control protocol *(DCCP)* with a new congestion control component. Extended DCCP is a transport layer protocol that implements the following functions: *1)* buffering of the received packets at the receivers, *2)* retransmission of the corrupted or lost packets by the sources, *3)* detection of duplicate packets at the receivers. Moreover, each sender node has four operation modes.

- *Normal state:* the senders adjust their rates based on the minimum recorded RTT over the average RTT.

- *Congestion state:* When a ECN is received, rate will be reduced similar to normal state if there aren't any packet losses. In case of packet losses the transmission rate will be halved.
- *Failure state:* probe packets are sent out to monitor the network situation.
- *Error state:* according to the state of error a new rate is calculate.

Fang et al. proposed *CADA*, congestion avoidance detection and alleviation in WSN [68]. CADA measures the congestion levels at each node by both queue occupancy and channel utilisation. A node will detect congestion if the queue occupancy exceeds the threshold or the channel utilisation reaches the maximum achievable channel utilisation. If congestion takes place in an intersection hotspot, a resource control mechanism will be applied. On the contrary if congestion is takes place in a convergence hotspot, a traffic control mechanism will be activated.

Antoniou et al. proposed Lotka-Volterra based congestion control *(LVCC)* in [69]. This study is mainly focusing on streaming applications in WSN and congestion prevention based on the Lotka-Volterra population model. LVCC requires minimum exchange of information and low computational burden.

2.4.3 Path Adaptation Schemes

The direct diffusion dissemination paradigm is proposed in [70]. In this study sink nodes announce their interest, which is a task description, to all sensors. The task descriptors are named by assigning attribute-value pairs describing the task. Each sensor node stores the received interest announcement on their cache. Each interest announcement entry contains a time-stamp and several other gradient fields. As the interest is propagated throughout the sensor network, the gradients from the source back to the sink are set up. When the source has data for the interest, the source sends the data along the interest's gradient path. The interest and data propagation and aggregation are determined locally. Also, the sink must refresh and reinforce the interest when it starts to receive data from the source. Note that the directed diffusion is based on data-centric routing where the sink broadcasts the interest.

A practical *resource control* scheme is proposed in [71]. When the congestion degree exceeds a predefined threshold, the algorithm in this scheme will wake up inactive nodes and calculate new alternative paths. With this approach the hotspots are bypassed by redirecting the traffic through the alternative paths. This scheme ignores the character of the centralised traffic patterns in WSN.

Wan et al. proposed Siphon [72]. In this work, the concept of virtual sinks *(VS)* is introduced. VS can be distributed dynamically for the tunneling of traffic events from areas with high sensor traffic. At the point of congestion, these VS divert extra traffic through them in order to maintain the required throughput at the base station. The Siphon algorithm mainly aims at addressing the VS discovery, operating scope control, CD, CA and traffic redirection. For the optimal operation of this protocol, VS nodes require dual radio. One for the low power communication between the nodes and a high power one for the communication between the VS.

BGR, a biased geographical routing protocol has been proposed in [73]. In BGR two congestion control algorithms namely: in-network packet scatter *(IPS)* and end-to-end packet scatter are used in order to avoid the direction of the packets through the congested areas of the network. IPS alleviates transient congestion by splitting the traffic before the congested areas. On the contrary, EPS alleviates long term congestion by splinting the flows at the source nodes. EPS select the paths dynamically and uses a less aggressive congestion control mechanism for energy efficiency.

Kang et al. proposed the topology aware resource adaptation *TARA* protocol in [74]. In TARA, congestion is detected by both queue occupancy and channel loading. The congestion alleviation in TARA, is performed with the assistance of two important nodes. These are the distributor and the merger. Between the important nodes, an alternative path is established. The distributor splits the traffic between the original and the alternative path and the merger merges the two flows. When there is no congestion in the network the alternative path is not used.

HTAP, a hierarchical tree alternative path protocol is proposed in [75]. HTAP is based on the creation of alternative paths from the source to the sink. In order to safely transmit data a creation of alternative paths which are not in the initial shortest path are created. These alternative paths are calculated by a hierarchical tree algorithm based on the congestion state in the network. When a node becomes congested, a back pressure message is transited. The receivers of the backpresure message will stop any active transmissions towards the congested nodes and immediately search their routing table for an alternative, less congested path.

A QoS adaptive congestion control scheme is proposed in [76]. In this study, each node calculates the packet service rate based on the processing time per packet at the MAC layer. According to the packet service rate the scale of congestion is estimated. There are two mechanisms implemented in this protocol for congestion control. The first one is called short term congestion control. With short term congestion control before the congested node, the traffic is split to its alternate parent in proportion to a weight factor. The second mechanism implemented in this protocol is called long term congestion control. In this mechanism, the congested node transmits a backpressure message. If the sources receive this message, traffic will be split between alternative paths from the source node.

He Tao et al. proposed *(TADR), TADR* a traffic aware, dynamic routing protocol. In this study, the proposed protocol, route packets around the congested areas by scattering the excessive packets through idle and underloaded nodes. TADR algorithm, constructs a mixed potential field using depth and normalised queue length. Based on this field, packets avoid the congested areas and eventually move towards the sink.

Hsu et al. proposed an adaptive NAV-assisted routing protocol *(ANAR), ANAR*. ANAR is a cross-layer implementation which encapsulates network allocation vector *(NAV)* information in the Request-To-Send *(RTS)* and Clear-To-Send *(CTS)* packets of the MAC protocol. NAV vectors are then used for the calculation of a congestion free probability which in turn carried through the route discovery process and based on this information, a feasible route for the packet delivery is determined. ANAR can dynamically switch between paths when the levels of congestion change.

A congestion avoidance and fairness protocol has been proposed in [77]. Each node with this protocol, will calculate the number of its upstream and downstream neighbours. The characteristic ratio *(CR)* is then measured as the ratio of the upstream over the downstream neighbours. If the value of CR is higher than 1, a node will forward packets towards the upstream node with the lower queue occupancy. If a node has more downstream than upstream neighbours, the value of CR will be lower than one. In that case, in order to prevent congestion a rate reduction algorithm will be activated and the transmission rates will be adjusted. This protocol assumes that a perfect- collision free MAC is in operation.

TALONet: a power efficient grid based congestion avoidance scheme is proposed at [78]. TALONet operate in three phases. *1) Network formation:* in this phase, information about the sinks location and the square grid *(virtual grid)* topology are transmitted from the sink towards every sensor node. Sensor nodes can then operate as TALON or normal nodes. A node will operate as a TALON node if its location is close to the virtual grids cross points. *2) Data dissemination:* during this phase, normal nodes forward the packets to their closest TALON node. A TALON node will forward the received packets to another TALON node until this information reach the sink. *3) Framework update:* In order to preserve energy, sink will periodically broadcast control packets including offsets for every node in the network which in turn will result in a new network formation.

A grid based multi-path with congestion avoidance routing protocol *(GMCAR)* has been proposed in [79]. GMCAR will form squared-shaped grids of predefined size. When the grids are created, a node will randomly selected to operate as the master node. Sink nodes will then flood the network with messages in order to discover routing paths from all the grids. All nodes in a grid will transmit their information to the master node and consequently, master nodes will forward this information to the sink. In order to avoid congestion, multiple diagonal paths are created between the master nodes and the sink. Each formed path is assigned with a weight. Traffic will then be distributed to the paths based on their weight.

DAIPaS: a dynamic alternative path selection algorithm is proposed in [80]. DAIPaS combines the information about nodes remaining power and congestion levels in order to calculate the optimal path between sources and sink. This protocol is operating in two phases: soft and hard. During the soft phase, nodes that serve more than one traffic flow will estimate which flow has the highest bandwidth and keep servicing. The rest of the flows will be forced in an alternative path. This algorithm can avoid the creation of hot spots during low traffic conditions. A node will enter the hard phase when traffic should not be routed through that node. Nodes in this phase will become unable to accept any more traffic. Nodes with queue occupancies close to the upper limit will temporarily enter hard phase. On the contrary, unavailable nodes or nodes with low power levels will permanently enter the hard phase.

2.4.4 Dynamic Duty Cycle Adaptation Schemes

S. H. Lee et al. propose AMAC: a traffic-adaptive sensor network MAC protocol through variable duty cycle operations [10]. In AMAC when the network is idle, Request to Send / Clear to Send *(RTS/CTS)* messages are not performed for energy saving. In order to achieve variable duty cycle operations, two basic components are used in this work: a) a clock synchronisation algorithm. Each node synchronising its wake-up time with its neighbours in a similar way to S-MAC. b) a schedule synchronisation mechanism. This mechanism is responsible for the synchronisation of wake-up times between neighbour nodes when AMAC nodes have adapted their cycle times.

J. Jeon propose DCA: A duty cycle adaptation algorithm for 802.15.4 beacon-enabled WSNs [11]. DCA assumes that beacon order *(BO)* is constant and adapts superframe order *(SO)* according to the superframe estimations. In this protocol, the control field in MAC protocol data unit *(MPDU)* is modified in order to gather traffic information from the end devices. After all the data are collected, the DCA coordinator estimates the number of packets being queued in the end devices and adjusts the SO accordingly.

S. Bac propose a traffic-aware MAC protocol using adaptive duty cycle for

WSN [12]. In this work, each node adjusts its duty cycle based on the traffic intensity measured by the queue length in its child nodes. A parent node will enter listen state and remain in this state until all the packets from its child nodes have been received. This protocol adds some additional overhead for the exchange of the traffic intensity between child and parent nodes. Furthermore, in order for this protocol to be functional, all packets in the WSN must be of the same size and traffic must be unidirectional *(from the edges of the network towards the sink)*.

N. Saxena propose a dynamic duty cycle and adaptive contention window based on QoS-MAC protocol for wireless multimedia sensor networks [13]. This protocol periodically measure the number of transmitted packets and calculates the probability of transmission failure based on the success-failure packet transmission ratio. When a probability of transmission failure is high a node will adjust its contention window *(CW)* and wait for its neighbour nodes to adjust their CW accordingly. Each node will keep adjusting its own CW until it achieves its CW-target. This protocol also suggests a scaling factor for the different traffic classes. The scaling factor is used by the CW increase and decrease algorithm for service differentiation between the different traffic classes. In this protocol, duty cycle times are also based on the class of the traffic. Depending on the dominating traffic class *(which traffic class had the most transmitted packets)*, the algorithm selects the active time and adjusts the duty cycle.

TA-MAC [14] is an adaptive duty cycle protocol for WSN. In TA-MAC, all nodes must be synchronised. In order to achieve node synchronisation, all nodes keep their radio on until they receive a SYNC packet. The SYNC packets is first broadcasted by the sink and then further propagated with broadcast messages by the rest of the nodes. In contrast to this protocols name, duty cycle is not altered under any network conditions; instead two or more packets may be transmitted in a single cycle through a two-way hand shaking mechanism *(DATA/ACK)*.

R.D.P. Alberola propose DCLA: a duty cycle learning algorithm for IEEE 802.15.4 beacon-enabled WSN [15]. In DCLA one node will operate as the coordinator. The coordinator node need to employ some estimation of the end devices traffic requirements in order to calculate an optimal duty cycle. In order to achieve this, the coordinator calculates the number of received messages during an active period. On the other side, end-devices embed their transmit queue occupation and delay values in the MAC header of sent data frames. When a coordinator node has no knowledge regarding the end devices the optimal duty cycle is calculated by the DCLA agent. The technique used for the calculation of the optimal duty cycle in that case is known as Q-Learning.

L. Dongho propose ADCC: an adaptive duty cycle based on congestion control scheme for home automation networks [16]. ADCC calculates the required service time based on the information received from the incoming packets, then makes a decision of whether this node is congested. Two mechanisms are implemented in this protocol in order to alleviate congestion. First, congestion is detected based on the required packet service time. After congestion is detected, each node will increase the duration of their active state based on the calculated congestion degree. Then a congestion notification message is broadcast by the congested node. When a node receives a congestion notification message, it will adjust its transmission rate. The rate reduction on a node is equal to the required service time over the maximum duty cycle active state in that node.

M. Anwander propose BEAM: a burst aware energy efficient adaptive MAC protocol for WSN [17]. The BEAM protocol is designed as an improvement upon X-MAC. The BEAM protocol comprises two different operational modes to optimise receiver sleep time dependent on the payload size. Both modes rely on positive acknowledgments of MAC frames upon reception. The two operational modes of this protocol are: *a)* basic operation mode, *b)* short preambles mode. In the first mode, the receivers wake-up and listen to the channel. If the receiver's address matches the address in the preamble an ack frame is transmitted. A sender node will continuously transmit a short preamble with payload frame until a positive acknowledgment is received. In the second mode, the receiver listens to the channel for preamble frames. If the address in the preamble matches the receiver's address an ack frame is transmitted. Upon the reception of an ack frame the sender is informed that the receiver is awake and transmits the data frame. BEAM's switching between these two states depends in the payload size *(for packets with more than 40 byte payload, basic operation mode is used)*. Furthermore, BEAM can transmit more than one frame to the same neighbour *(packets can be aggregated)* if there is enough space for both packets in a single MAC frame. During a strong traffic increase, BEAM uses 1 bit information *(traffic indicator)* in the frame control field *(FCF)* of every transmitted frame in order to inform its parent node about the traffic increase. Upon the reception of a frame with the traffic indicator bit set, a node will adapt its listen cycle by calculating an earlier time to wake-up.

H. Hu et al. propose ADC-SMAC [81]: an improvement of S-MAC based on dynamic duty cycle. In ADC-SMAC, each node periodically calculate its utilisation and sleeping delay. This information is then used for the calculation of a new duty cycle. After the calculation of the new duty cycle, nodes will share the information with their neighbours. In ADC-SMAC the information sharing between neighbour nodes is performed through the transmission of broadcast frames.

H. Yoo et al. propose DSR [82]: duty cycle scheduling based on residual energy. In this mechanism, each sensor calculates it's residual energy every time it wakes up. The duty cycle at each node is then calculated based on the residual energy. The algorithm used for the duty cycle calculation in DSR can is:

$$I_{dc}^{i} = I_{dc}^{max} - (I_{dc}^{max} \times \frac{E_r^i - E_{th}}{E_{max} - E_{th}}) \tag{2.1}$$

were I_{dc}^{i} is the current duty cycle, I_{dc}^{max} is the maximum duty cycle, E_r^i is the nodes residual energy, E_{max} is the maximum residual energy and E_{th} is the residual energy threshold.

2.4.5 A Detailed Comparison of Congestion Control Schemes

In previous sections a categorisation of various CC, based on the CA mechanisms each scheme has embodied, was introduced. In this section, a more generic analysis will be made in which all schemes described previously will be evaluated as a whole. Additionally, an individual analysis of the mechanisms incorporated by the various schemes will be presented. Table 2.1 shows all 57 CC approaches discussed in this thesis as well as the individual mechanisms each scheme uses for CA,CD and Congestion Notification *(CN)*. The protocols in Table 2.1 are shorted based on a chronological order. In subsection 2.3.1, a categorisation of the CC schemes based on their CA mechanisms was introduced and thus the individual techniques for CA hve already been described in detail. On the contrary CD and CN have not been described in such detail and therefore during the rest of this Section CD and CN mechanisms will also be categorised and described in detail in order to present a complete analysis on CC for WSN.

- *Congestion Detection(CD):* is usually referred to mechanisms that try to predict if congestion is going to occur in advance. The most common methods for CD are:

  1. *Queue Occupancy:* nodes monitor their local queue. When the queue occupancy exceed a predefined threshold congestion is detected.
  2. *Channel Status:* includes collisions and channel load or traffic rate. When the channel load increase dramatically congestion is detected. Some schemes will detect congestion and reduce their transmission windows based on the collisions occurred.
  3. *Failed Transmissions:* usually when a packet does not get acknowledged congestion is detected. The type of the acknowledgment varies based on the network layer each scheme is implemented at *(usually link, network and application layer)*.

4. *Inter Packet Arrival Over Service Time:* nodes measure their local packet service time *(P_s)* and packet arrival time *(P_a)*. If the ratio of P_s/P_a is lower than one, packets are arriving faster than the node can service and thus congestion is detected.

5. *Energy usage:* includes techniques such as power used for the packet transmission or total energy consumed by the radio. If the energy consumption due to packet transmission/reception in a node exceed an energy threshold congestion is detected. In addition some protocols modify the transmission energy in the radio in order to reduce the transmission power and thus the interference and packet collision frequency.

- *Congestion Notification(CN):* usually refers to mechanisms used by the various schemes in order to exchange information between the nodes.

  1. *Explicit Congestion Notification (ECN):* non data packets are explicitly used for the exchange of congestion information between the nodes.

  2. *Implicit Congestion Notification (ICN):* congestion information exchanged between the nodes are encapsulated in the data packets.

Table 2.1: Congestion control schemes

Protocols	Congestion Detection	Congestion Avoidance	Congestion Notification	Category &Year
CODA [83]	Queue occupancy & channel status	AIMD end-to-end & hop-by-hop traffic control	Explicit	Rate adaptation *(2003)*
ESRT [46]	Queue occupancy	Reliability based end-to-end traffic control	Explicit	Rate adaptation *(2003)*
Direct diffusion [70]	Failed transmissions	Traffic redirection	N/A	Path adaptation *(2003)*
Application based collision avoidance [37]	Failed transmissions	AIMD end-to-end traffic control	Explicit	Collision avoidance *(2004)*
SRBP ARBP and RARBP [38]	Collision occurrence	CSMA Back-off retransmission	N/A	Collision avoidance *(2004)*
Hybrid collision avoidance [39]	Unsuccessful RTS packets	RTS/CTS phase not performed	Implicit	Collision avoidance *(2004)*
Fusion [47]	Queue occupancy & channel status	Start and stop hop-by-hop traffic control	Implicit	Rate adaptation *(2004)*

CCF [48]	Queue occupancy	Phase shifting & hop-by-hop traffic control	Implicit	Rate adaptation *(2004)*
Resource control scheme [71]	Packet inter arrival and packet service time	Traffic redirection	N/A	Path adaptation *(2004)*
Carrier sence [40]	Energy detect from signal received	CSMA Back-off retransmission	N/A	Collision avoidance *(2005)*
Idle sence [41]	Failed transmission at MAC layer	AIMD increments of CSMA Back-off window	N/A	Collision avoidance *(2005)*
RBC [49]	N/A	Virtual queues and prioritised packet transmission	Implicit	Rate adaptation *(2005)*
STCP [50]	Queue occupancy	AIMD end-to-end traffic control	Explicit	Rate adaptation *(2005)*
SenTCP [51]	Queue occupancy & packet inter arrival time	Hop-by-hop traffic control	Implicit	Rate adaptation *(2005)*
Siphon [72]	Queue occupancy & Channel status	Traffic redirection	N/A	Path adaptation *(2005)*
AFA [52]	Queue occupancy	Start and stop Hop-by-hop traffic control	Implicit	Rate adaptation *(2006)*
IFRC [53]	Queue occupancy	AIMD hop-by-hop traffic control	Implicit	Rate adaptation *(2006)*
Lightweight buffer management [54]	Queue occupancy	Start and stop hop-by-hop traffic control	Implicit	Rate adaptation *(2006)*
BGR [73]	Queue occupancy & Channel status	Traffic redirection & hop-by-hop traffic control	Implicit	Path adaptation *(2006)*
CSMA/PB [84]	Collision occurrence	Power back-off *(reduce transmission power)*	N/A	Collision avoidance *(2007*
E-CSMA [42]	Ratio of failed packet transmissions	CSMA Back-off retransmission	Implicit	Collision avoidance *(2007)*
Contention access for collision avoidance [43]	Collision occurrence	50% reduction of the nodes to wake up	N/A	Collision avoidance *(2007)*
PCCP [55]	Packet inter arrival and packet service time	Hop-by-hop traffic control	Implicit	Rate adaptation *(2007)*

DPCC [56]	Queue occupancy & channel status	Hop-by-hop traffic control	Explicit	Rate adaptation *(2007)*
RCRT [57]	Based on time to recover packet loss	AIMD End-to-end traffic control	Explicit	Rate adaptation *(2007)*
HRCCP [58]	Queue occupancy, timer & packet error	Hop-by-hop traffic control	Explicit	Rate adaptation *(2007)*
ABPS [59]	Queue occupancy	Hop-by-hop traffic control	Explicit	Rate adaptation *(2007)*
TARA [74]	Queue occupancy & Channel status	Traffic redirection	Explicit	Path adaptation *(2007)*
HTAP [75]	Queue occupancy	Traffic redirection	Explicit	Path adaptation *(2007)*
AMAC [10]	Channel status	Duty cycle adaptation *(requires node syncronisation)*	Explicit	Dynamic duty cycle *(2007)*
DCA [11]	Queue occupancy	Duty cycle adaptation *(requires node syncronisation)*	Implicit	Dynamic duty cycle *(2007)*
HCCP [60]	Queue occupancy & packet inter arrival and packet service time	Hop-by-hop traffic control	Explicit	Rate adaptation *(2008)*
COMUT [61]	Cluster/traffic intensity	Hop-by-hop *(cluster by cluster)* traffic control	Explicit	Rate adaptation *(2008)*
QCCP-PS [63]	Queue occupancy	Hop-by-hop traffic control	Explicit	Rate adaptation *(2008)*
QoS adaptive congestion control [76]	Packet inter arrival and packet service time	Traffic redirection	Implicit	Path adaptation *(2008)*
TADR [85]	Queue occupancy	Traffic redirection	N/A	Path adaptation *(2008)*
ANAR [86]	Channel status	Traffic redirection	Implicit	Path adaptation *(2008)*
Congestion avoidance and fairness [77]	Queue occupancy	Traffic redirection & Hop-by-hop traffic control	Implicit or Explicit	Path adaptation *(2008)*
Traffic aware MAC [12]	Queue occupancy	Duty cycle adaptation *(requires node syncronisation)*	Explicit	Dynamic duty cycle *(2008)*

Dynamic duty cycle and adaptive contention window based QoS MAC [13]	Successful over failed packet ratio	Duty cycle adaptation *(requires node syncronisation)* & traffic control	N/A	Dynamic duty cycle *(2008)*
TA-MAC [14]	N/A	Duty cycle adaptation *(requires node syncronisation)*	Implicit	Dynamic duty cycle *(2008)*
Inverting wireless collision [44]	N/A	Start and stop transmission time at neighbour nodes	Implicit	Collision avoidance *(2009)*
FACC [64]	Channel status	Hop-by-hop traffic control	Implicit	Rate adaptation *(2009)*
UHCC [65]	Queue occupancy & Channel status	Hop-by-hop traffic control	Implicit	Rate adaptation *(2009)*
LACAS [66]	N/A	Learning automata traffic control	N/A	Rate adaptation *(2009)*
Extended-DCCP [87]	Packet loss similar to *FAST-TCP*	End-to-end AIMD traffic control	Explicit	Rate adaptation *(2009)*
CADA [68]	Queue occupancy & Channel status	Hop-by-hop traffic control	Implicit	Rate adaptation *(2009)*
LVCC [69]	Queue occupancy	Hop-by-hop traffic control	Implicit	Rate adaptation *(2009)*
TALONet [78]	Queue occupancy	Traffic redirection & traffic control	N/A	Path adaptation *(2009)*
GMCAR [79]	Queue occupancy	Traffic redirection	N/A	Path adaptation *(2009)*
IBPS [45]	N/A	Transmission back-off at the nodes interfering with the sender	Explicit	Collision avoidance *(2010)*
DCLA [15]	Queue occupancy & Channel status	Duty cycle adaptation *(requires node syncronisation)*	Implicit *(congestion information)* & Explicit *(DC adaptation)*	Dynamic duty cycle *(2010)*
ADCC [16]	Packet service time	Duty cycle adaptation & traffic control	Explicit	Dynamic duty cycle *(2010)*
BEAM [17]	Channel status	Duty cycle adaptation	Implicit	Dynamic duty cycle *(2010)*

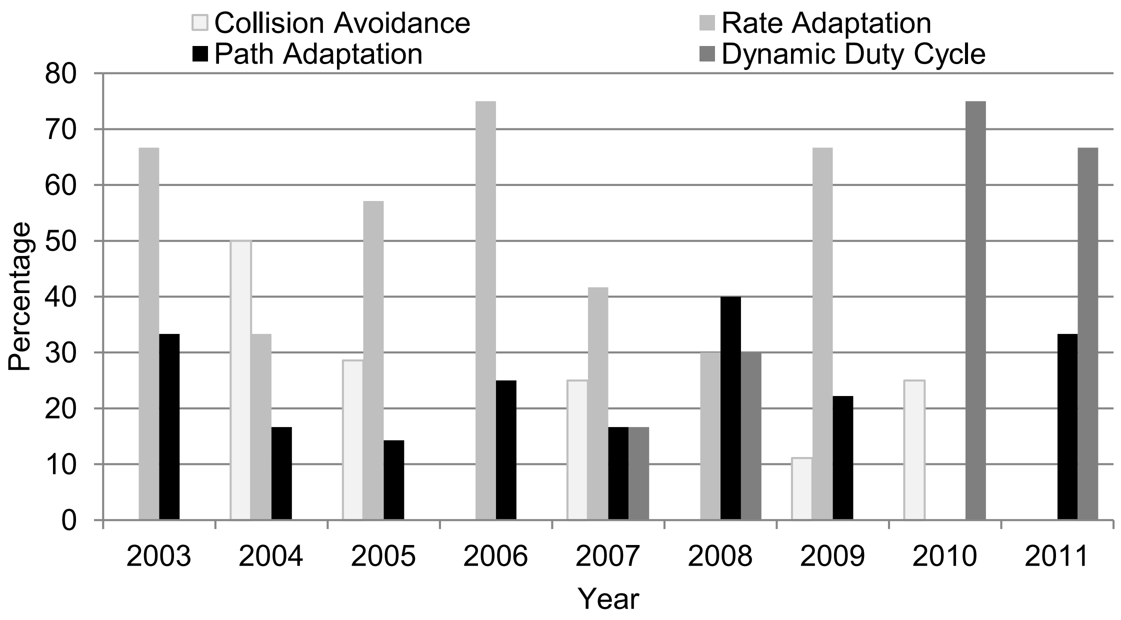

Figure 2.3: Percentage of the overall congestion research field over the years for the different categories

DAIPAS [80]	Queue occupancy	Traffic redirection	N/A	Path adaptation *(2011)*
ADC-SMAC [81]	Channel status & sleeping delay	Duty cycle adaptation	Implicit	Dynamic duty cycle *(2011)*
DSR [82]	Residual energy	Duty cycle adaptation	N/A	Dynamic duty cycle *(2011)*

According to the scheme categorisation proposed in subsection 2.3.1, Table 2.2 describes how many schemes and from which category were suggested during the last years. In addition, Figure 2.3 represents the percentage of the number of different categories schemes over the total schemes proposed. Further studying the above table, it can be observed that rate adaptation schemes were the most popular choice for congestion control within the WSN research community. It is also noticeable that overall, the number of rate adaptation schemes proposed by WSN research community is approximately double from the second most popular category which is the path adaptation schemes.

Moreover, there was no dynamic duty cycle protocols proposed before 2007. Additionally, based on Figure 2.3 dynamic duty cycle schemes have attracted lots of attention from the WSN community since their appearance with significantly higher numbers of schemes developed in the late years compared to the rest of the categories.

Table 2.2: Congestion control categories over the years

Categories	2003	2004	2005	2006	2007	2008	2009	2010	2011	Total
Collision Avoidance	0	3	2	0	3	0	1	1	0	10
Rate Adaptation	2	2	4	3	5	3	6	0	0	25
Path Adaptation	1	1	1	1	2	4	2	0	1	13
Dynamic Duty Cycle	0	0	0	0	2	3	0	3	2	10
Total	3	6	7	4	12	10	9	4	3	58

In fusion [47], authors suggested that queue occupancy is sufficient for CD. On the contrary, others [60, 65, 68] suggest that queue occupancy is not sufficient.Table 2.3 demonstrates the use of various CD mechanisms through the years. Based on the above table, it is easy to conclude that measuring the queue is the most common approach for CD in WSN and it is used by the majority of CC schemes.

Table 2.4 demonstrates which CD mechanisms are used by the different CC scheme categories. Through this table, it is visible that the majority of rate adaptation and path adaptation schemes use the queue occupancy in order to detect congestion. On the contrary, collision detection schemes mainly use the transmission status as a congestion indicator. This is expected since collisions and the hiden-terminal problem *(collision at the receiver)* are usually the reason behind failed transmission in WSN *(failed packet transmissions can also be caused due to corrupted data or link failure but these are not related to congestion)*. It is also observed that all of the CD mechanisms have been used in the design of dynamic duty cycle schemes.

Table 2.3: Congestion detection mechanisms over the years

Congestion Detection Mechanism	2003	2004	2005	2006	2007	2008	2009	2010	2011	Total
Queue Occupancy	2	2	4	4	6	5	5	1	1	30
Channel Status	1	1	1	1	3	1	3	3	1	15
Failed Transmissions	1	3	1	0	5	1	1	0	0	12
Inter Packet Arrival Over Service Time	0	0	2	0	1	2	0	1	0	6
Energy	0	0	1	0	0	0	0	0	1	2
N/A	0	0	1	0	0	1	1	1	0	4

Table 2.4: Congestion detection mechanisms for the different CD categories

	Collision Avoidance	Rate Adaptation	Path Adaptation	Dynamic Duty Cycle
Queue Occupancy	0	18	9	3
Channel Status	1	5	5	4
Failed Transmissions	7	3	1	1
Inter Packet Arrival Over Service Time	0	4	1	1
Energy	1	0	0	1
N/A	1	2	0	1

Studying Table 2.5 and Table 2.6, it is clear that ICN is the most preferred method for congestion knowledge sharing between the nodes. As mentioned previously, with ICN the information is encapsulated in the data packets and thus no additional overhead is inserted in the network. During congestion, the network is heavily loaded; therefore avoiding any extra packet transmissions and successfully sharing the information is the ideal solution. This can easily explain why ICN is the most preferred approach for congestion sharing by the CC schemes.

Table 2.5: Congestion sharing approaches over the years

Congestion Notification	2003	2004	2005	2006	2007	2008	2009	2010	2011	Total
Explicit	2	1	1	0	7	5	1	0	0	17
Implicit	0	3	3	4	3	4	5	2	1	25
N/A	1	2	3	0	2	2	3	2	2	17

Table 2.6: Congestion sharing approaches for the different categories

Congestion Notification	Collision Avoidance	Rate Adaptation	Path Adaptation	Dynamic Duty Cycle
Explicit	2	11	3	1
Implicit	3	13	4	1
N/A	5	1	7	2

Over all, there is no ideal CD, CA and CN approach. Based on the networks traffic patterns various combinations of CC mechanisms may demonstrate different results. In order to explain the above statement lets assume the following two scenarios:

1. an end-to-end approach will demonstrate better results by using ECN for CN and rate or path adaptation for CA since the source is directly communicating with the sink and there is no need for a high degree of congestion information at the intermediate nodes.

2. a hop-by-hop approach that tries to mitigate or eliminate congestion can save lots of bandwidth combined with ICN instead of ECN.

2.5 Summary and Discussions

Subsection 2.2.1,and subsection 2.2.2 describe the importance of IP in WSN. Obviously IPv6 over 6LoWPAN has received lots of attention from researchers since architectures such as the above are of great importance for the interconnection of every device to the Internet *(Internet of Things)*. Additionally new routing protocols are tailored in the specific needs and characteristics of WSN. Subsection 2.2.3 describes the importance of new routing protocols for WSN. Subsection 2.2.4 introduces the duty cycle algorithms and describes how vital these algorithms are for the operation and lifetime of a sensor network. Through the rest of this review the importance of congestion control in sensor networks is demonstrated. Numerous approaches are attempting to solve the problem of congestion in all of its forms *(collision, buffer overflow)*. One of the most common approaches for congestion control is the traffic or rate control while different approaches that attempt to eliminate collision or diverge the traffic from the congestion hot-spots have also attracted lots of attention from the researchers. More recently, new methods that adapt a nodes duty cycle in order to confront congestion have been introduced. It is also observed that these approaches are becoming the focus of the WSN research community. A new categorisation of the existing congestion control schemes based on the congestion avoidance mechanisms incorporated by each protocol has been introduced in section 2.3. Details about the existing congestion control schemes for WSN can be found in section 2.4 while a summarisation of them is presented in Table 2.1. Table 2.6, Table 2.5, Table 2.4, Table 2.3 and Table 2.2. These tables analyse in detail the mechanisms incorporated by each congestion control scheme as well as the focus and direction of the research community in the past years.

Collision avoidance is very important for congestion control to combat the adverse effects of hidden terminals and collisions. Schemes focusing on collision avoidance, usually attempt to minimise simultaneous transmissions between interfering nodes. In order to achieve this, numerous mechanisms such as backing-off in time or in space have been proposed. In the former, nodes adapt their transmission windows and retransmit packets after random intervals when a collision is detected. The latter attempts to reduce interference between nodes by reducing the transmission range at the nodes. Reducing the frequency of radio collisions does not necessarily mean that the congestion problem has been fully resolved. Congestion can still occur, since the local fairness achieved by CSMA and collision avoidance protocols contributes to potential buffer overflows.

Rate adaptation approaches have received lots of attention for the confrontation of congestion. In general, sources know an explicit rate at which they can send. The rate may be given to the source during a negotiation phase or be specific to the application. End-to-end rate adaptation schemes usually attempt to dynamically impose the transmission rate based on the congestion state at the network. Alternatively, some rate adaptation approaches attempt to mitigate congestion from the intermediate nodes towards the sources. These schemes are usually piggyback in hop-by-hop fashion the congestion information and adapt the transmission rates at the MAC layer. Rate adaptation schemes can confront congestion but don't always eliminate it with success.

Architects of path adaptation schemes, believe that reducing the traffic during congestion is undesirable since it can violate fidelity requirements. This has led to the development of protocols able to increase the capacity of the network by utilizing a greater number of resources. Taking advantage of the nature of WSN and their usually dense deployment, path adaptation schemes are calculating numerous paths between the sources and the destination. When congestion occurs, the traffic is redirected towards an alternative path and thus congestion is confronted without any traffic reduction. These schemes usually calculate the alternative paths in advance and thus the lossy link nature of WSN may lead to undesirable behaviors or frequent recalculation of alternative paths and therefore unnecessary energy consumption. Moreover, these algorithms are usually tailored for specific traffic characteristics and thus their functionality over IPv6 sensor networks with diverse traffic patterns is uncertain.

Self powered WSN usually use a duty cycle algorithm in order to preserve energy and extend its lifetime *(it is not realistic to assume that a self powered sensor network operates without a duty cycle algorithm).* When congestion is detected, dynamic duty cycle schemes will increase the networks bandwidth by increasing the amount of time a node spends in the up state *(during the up state a node has its radio transceiver on).* The majority of these protocols require node synchronisation for the duty cycles reconfiguration.

In literature, there are a plethora of schemes, mechanisms and approaches for the confrontation of congestion. Even though there is a vast amount of research in the field of congestion, the majority of the existing congestion control schemes have not considered carefully the existence of an underlying duty cycle algorithm in the network *(except the dynamic duty cycle protocols).* Additionally, most of the existing protocols are tailored in the needs of semi-closed protocol stacks. Both duty cycling and IP architectures for sensor networks are becoming increasingly popular amongst the WSN research community. This, combined with the inadequate performance of TCP in WSN has lead to the need of new duty cycle aware congestion control protocols tailored in the unique characteristics of IP and 6LoWPAN.

Chapter 3

Experimental Setup and Empirical Practice

3.1 Introduction

This chapter presents a comprehensive analysis of the experimental setup and methodology used during the experiments. The Contiki OS, an open source operating system for the Internet of things will be described in section 3.2 while Cooja a network simulator for Contiki OS will be presented in section 3.3. IEEE 802.15.4 has become the standard of choice for low-rate wireless personal area networks (LR-WPANs) and used by the majority of the nowadays sensor networks. 802.15.4 and related physical layer and MAC specifications will be discussed in section 3.5. Radio duty cycle has attracted lots of attention by the WSN research community. Additionaly, understanding RDC's importance and applicability is vital for this thesis. Radio duty cycle and related algorithms implemented in Contiki OS will be described in detail in section 3.6. In order to apply the Internet Protocol to small devices with limited processing capabilities, 6LoWPAN has been introduced. Therefore, 6loWPAN constitutes an important role towards the Internet of things. 6LoWPAN will be discussed in section 3.7. uIP *(micro IP)* stack and related IP stacks for microcontrollers will be summarised in section 3.8 while Routing for Low Power and Lossy Networks *(RPL)* is discussed in section 3.9. Experiences on how to set up a real hardware test-bed and the methodology used for the experiments in this thesis will be explained in detail in section 3.10. Moreover, in order to implement a fully functional testbed, it is very important to collect the necessary data and store them in an easy to process format. Finally, the nodes and the data collection server should be able to adapt to various network parameters such as source rates and traffic patterns in order to avoid frequent reconfiguration of the nodes; which in turn can lead to wasted time and inconsistencies between the experiments. Details about the data collection server, the sensor applications and how auto network configuration was performed will be described in detail during section 3.11.

3.2 Contiki Operating System

Contiki is a highly portable multitasking computer operating system developed for use on networked embedded systems and wireless sensor networks and it is designed for microcontrollers with small amounts of memory. In general, the Contiki OS allows small devices such as wireless sensors to communicate with the Internet and each other with very low energy consumption. The main innovation of Contiki, is it's ability to allow resource-constrained systems to communicate using Internet protocol *(IPv4 and IPv6)*. Contiki provides a built-in TCP/IP stack. Even though, a typical configuration of Contiki OS can be as small as 2 kilobytes of RAM and 40 kilobytes of ROM. The Contiki team currently has members from Cisco, Redwire LLC, SAP, SICS, and other [24, 25, 26, 27].

The contiki OS, uses multiple communication stacks such as Rime and uIP/uIP6. It is a lightweight communication stack designed for low-power radios. Rime provides a wide range of communication primitives, from best-effort local area broadcast, to reliable multi-hop bulk data flooding [88, 89, 90]. The uIP embedded IP stack [91, 92, 93, 94] is currently used by hundreds of companies in systems such as freighter ships, satellites and oil drilling equipment and it is recognised by popular network scanning tools such as nmap and Wireshark. More details about the uIP is going to be presented in section 3.8.

Contiki, also uses an intuitive way of controlling multiple tasks through protothreads. Protothreads are a novel programming abstraction that provide a conditional blocking wait statement which intends to simplify event-driven programming for memory-constrained embedded systems [95]. Nowadays protothreads are used in many different places such as TV decoders, wireless vibration sensors and even games.

As of the time this thesis was written, the latest commercial version of Contiki OS was the 2.5 release. During this study, the latest version of Contiki OS *(frequently updated with the latest changes through Git)*; was used for the simulation and test-bed experiments.

3.2.1 Contiki Structure

Figure 3.1 demonstrates the structure of Contiki OS. In Contiki, applications and platform specific drivers can be developed independent of the core components.

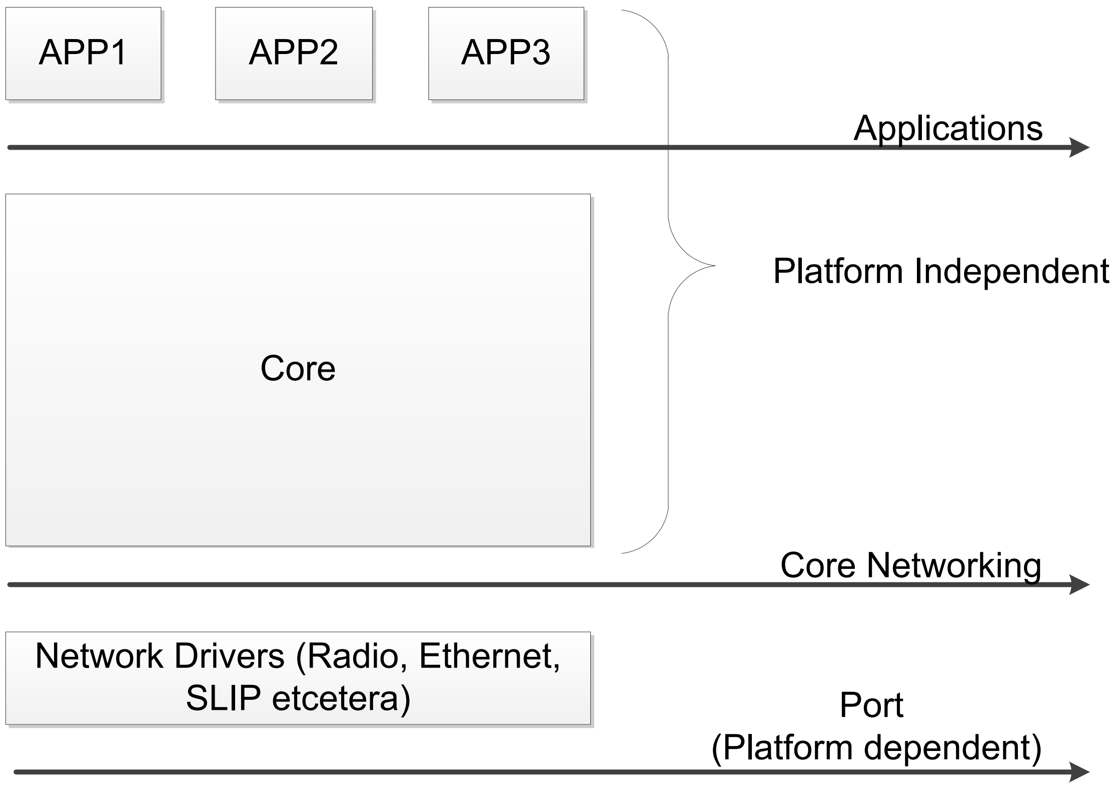

Figure 3.1: Structure of Contiki

Being designed this way Contiki is highly portable since various sensor and embedded system manufacturers can port their hardware independently of each other. Contiki also allows for communication between different hardware devices. Similarly to the drivers, applications are not part of the core and thus various nodes in the same network can incorporate different applications and communicate with each other at the same time.

The Contiki Core host the network, MAC and RDC layers as shown in Figure 3.2.

As mentioned earlier in this Chapter, Contiki uses multiple communication stacks. A detailed representation of the communication stacks as well as the rout-

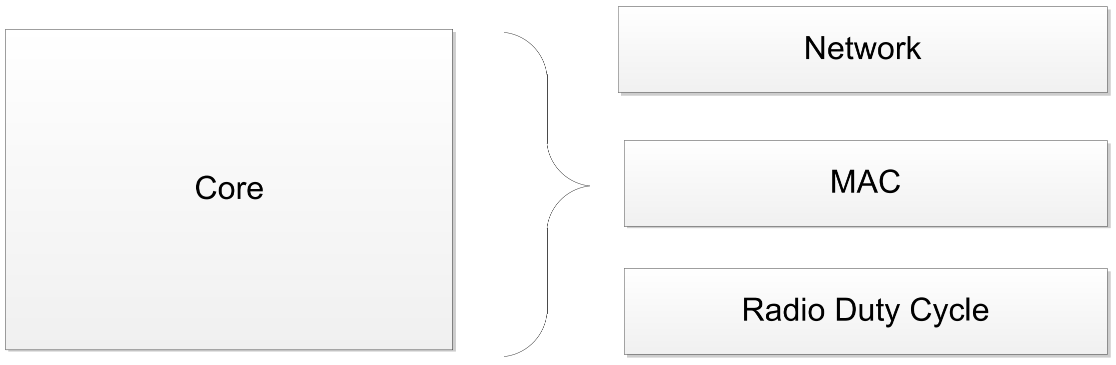

Figure 3.2: The Contiki core

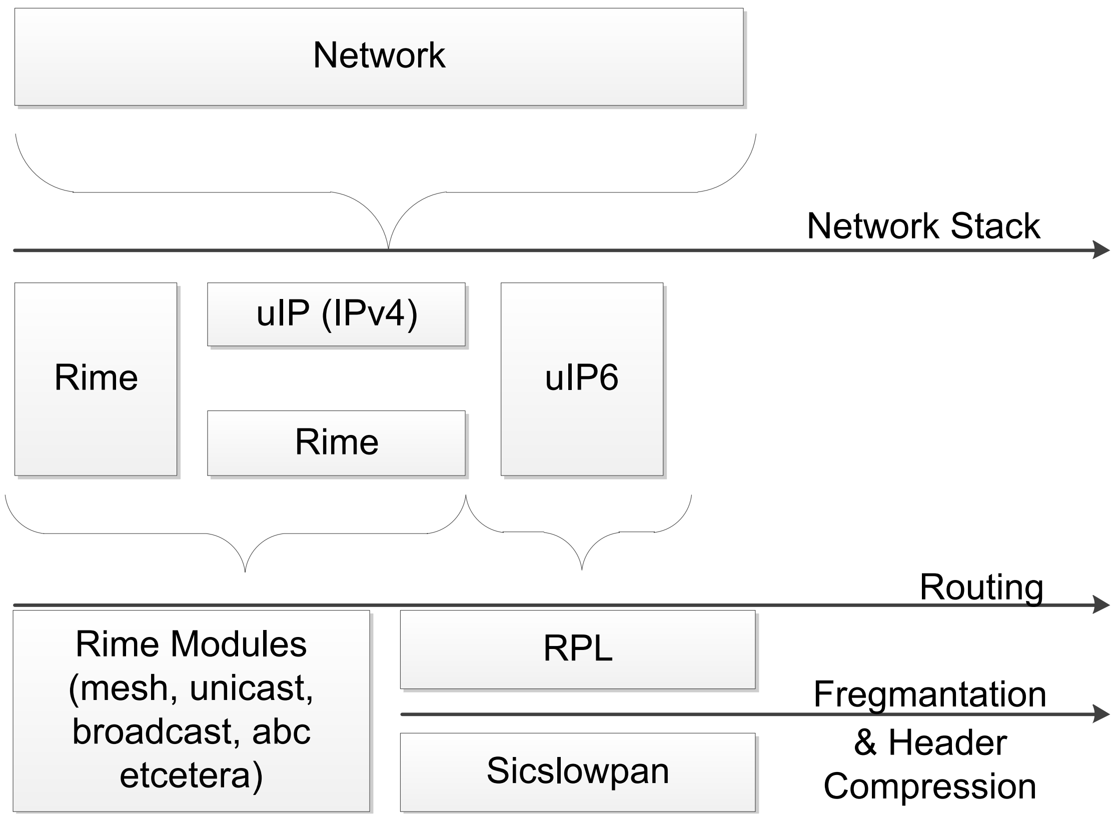
Network
Network Stack
Rime
uIP (IPv4)
Rime
uIP6
Routing
Rime Modules (mesh, unicast, broadcast, abc etcetera)
RPL
Fregmantation & Header Compression
Sicslowpan

Figure 3.3: Networking in Contiki

ing and compression protocols in Contiki can be found in Figure 3.3. Contiki can be configured to use Rime, uIP or a combination thereof. An example of a combined use of Rime with uIP is a network configuration with IPv4. To be more precise, IPv4 in Contiki uses a mesh-under configuration in order to achieve mesh routing and route discovery [89]. This is done by the Rime communication stack. On the contrary, IPv6 in Contiki uses a route-over configuration. Therefore, in uIP6 routing is handled by RPL: *Routing Protocol for LLNs (currently only RPL protocol is implemented for IPv6 routing in Contiki)*. Additionally, header compression is provided by the 6LoWPAN module. The uIP and rime communication stack is going to be analysed in detail in section 3.8. RPL routing protocol and the 6LoWPAN module will be summarised in section 3.9 and section 3.7 respectively.

Figure 3.4 shows the structure of the MAC and RDC layer in Contiki as well as the existing protocol implementations in these network layers. In some OS for sensor networks such as Tiny OS, the MAC layer includes both the MAC and the radio duty cycling algorithms. In Contiki OS the RDC algorithms have been separated from the MAC *(implemented as a different network layer)* in order to increase the configuration flexibility. For example, CSMA MAC can be configured either with or without a duty cycle algorithm with no changes in the protocol or any need of alternative implementations of the same algorithm. Currently, Contiki

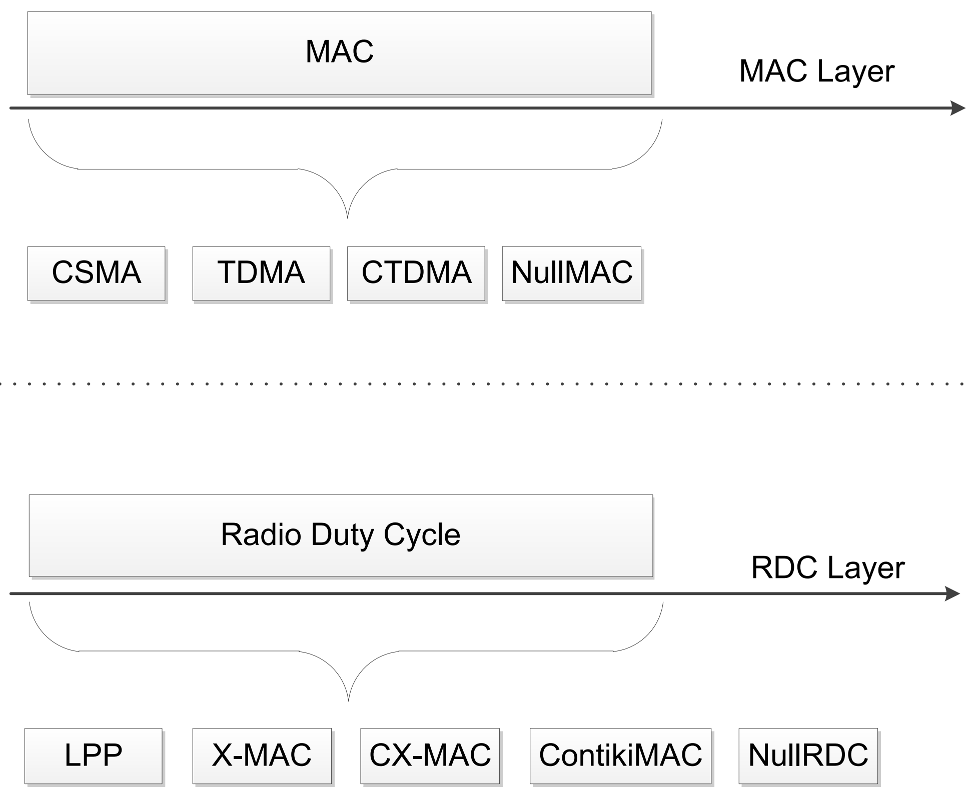

Figure 3.4: MAC and RDC layers in Contiki

can be configured with CSMA, TDMA, CTDMA *(same as TDMA implementation but uses different timers)* or NullMAC as its MAC layer and LPP, X-MAC, CX-MAC *(same as X-MAC implementation but uses different timers)*, ContikiMAC or NullRDC. The protocols mentioned above will be described in detail in section 3.6. The complete Contiki uIP6 network stack is presented in Figure 3.5

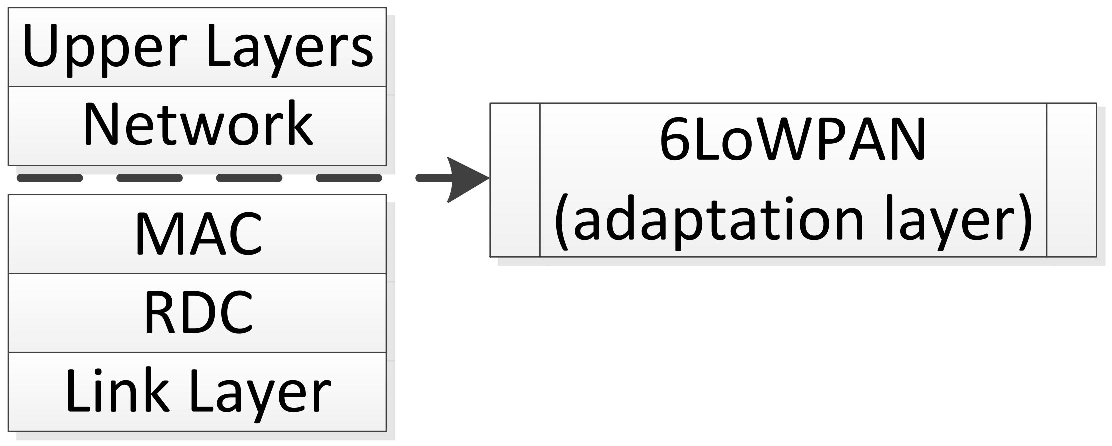

Figure 3.5: Contiki with uIP6 network stack

3.2.2 Why Contiki?

There are many alternative OS's to Contiki for sensor networks. The unique characteristics of Contiki and thus the reason behind the choice of this OS for the evaluation of the proposed algorithms, follows [96]:

- Contiki is an open source operating system.
- Out of the box data collection: Contiki has tools that allows the user to monitor the data collection in real time through a GUI.
- Extensive simulator support: Contiki has a full featured simulator suite that allows rapid testing of software for functional correctness *(The same codes used for simulations can be transferred to hardware devices without code modifications).*
- Good support: Contiki OS currently supports over 25 platform ports. Additionally, the OS is well documented and developers can get in touch directly through a mailing list and more recently an IRC channel.
- Contiki networking: Contiki has the first and smallest IPv6 compatible network stack and allows easy access and direct modifications to the network stack.
- Coffee Filesystem: a simple and easy to use filesystem that allows a user to store data without worrying about underlying problems.
- Power profiling: Contiki allows the user to monitor the energy usage with the energest library that can easily be extended based on the user needs.
- Extremely low power consumption: When Contiki is configured to duty cycle with ContikiMAC, nodes can achieve up to 99% lower energy consumption than non duty cycle configurations.
- Simple and straightforward: Contiki is written in simple and straight C99 *(C programming language standard).*
- Contiki Protothreads: Contiki uses the most intuitive way of controlling multiple tasks through protothreads, a very lightweight form of POSIX inspired threading [97, 98].

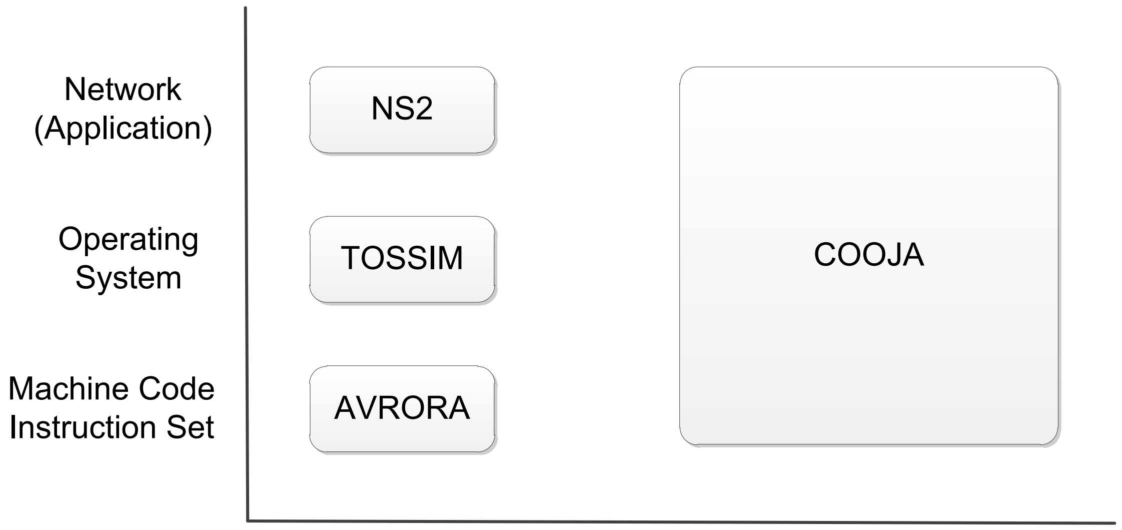

Figure 3.6: COOJA can simulate at several levels [2]

3.3 COOJA Simulator

Software development for sensor networks can be simplified when the developer uses tools such as network simulators. With network simulators, developers can study the system behavior and observe interactions in controlled environment [99, 100, 101, 102, 103]. The majority of existing network simulators for WSN, perform simulations either at the operating system or hardware level. The level at which the simulation is performed can affect both the software development and the execution efficiency of the simulator. In general, hardware level simulators can produce more accurate results and informations about the low-level software such as device drivers but at the price of longer simulation times and code complexity. On the contrary, a high-level simulator can provide short simulation times but does not provide any hardware level information.

COOJA is a network simulator for Contiki that enables cross-level simulation: simultaneous simulation at many levels of the system [104, 2]. Figure 3.6 demonstrates the operation levels *(can individually or simultaneously simulate nodes of application, operating system or machine code level)* of the COOJA simulator compared to other simulators.

COOJA is implemented in Java and thus it is very easy for the users to extend it, while it allows sensor node software to be written in C by using the Java Native Interface *(JNI)*. Furthermore, COOJA is flexible and allows many parts of the simulator to be replaced with ease. This way users may develop their own modules such as visualiser plugins and radio modules in order to enhance COOJA and achieve the additional functionality required.

Moreover, COOJA simulates networks of sensor nodes where each node can be of a different type; both in on-board software and in the simulated hardware.

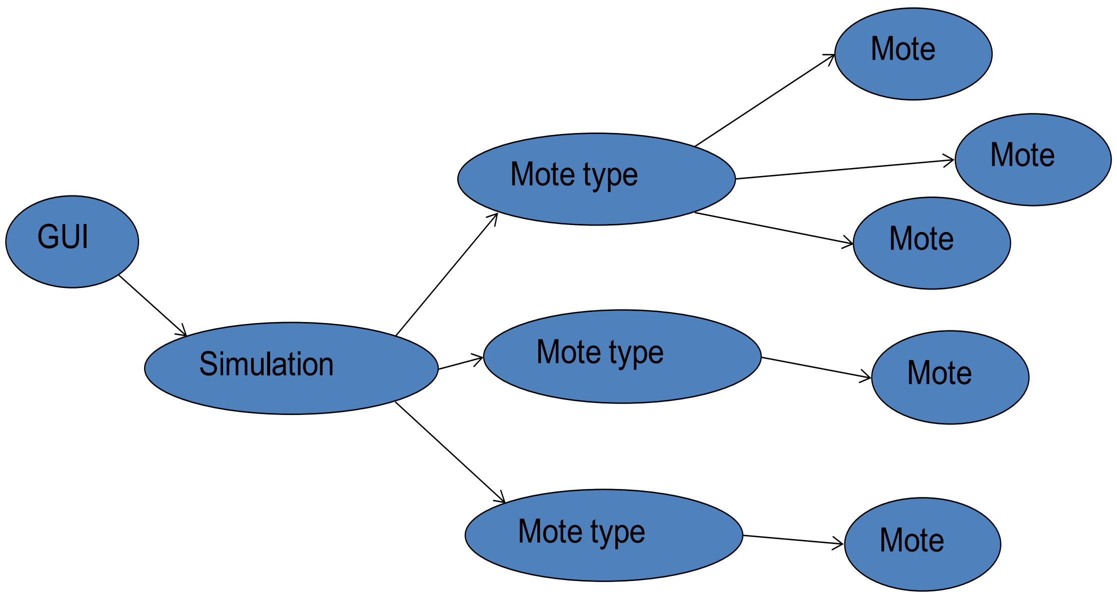

Figure 3.7: Example of various level nodes in COOJA [2]

All nodes simulated by COOJA have 3 basic properties: data memory, node type and hardware peripherals. Since COOJA is a cross-layer simulator, nodes can be simulated at multiple layers. Therefore, COOJA supports three different categories of nodes: Application or java nodes, native or OS level nodes and emulated or machine code nodes. An example of how COOJA can simulate various level nodes can be seen in Figure 3.7.

The three different categories of simulated nodes and their different pros and cons are listed below:

Application or java nodes: much faster simulations, can not implement deployable code.

Native or OS level nodes: more efficient than application nodes, can simulate deployable code.

Emulated or machine code nodes: provide more fine-grained details compared to the other categories but much slower simulation times.

All in all, COOJA is a very useful network simulator and it is recommended for testing sensor network applications written for Contiki. The users can simulate nodes in various layers including hardware and produce fine-grained details about their algorithms.

3.4 Sensinode and the CC2431 System-on-Chip

The devices used for the implementation of our testbed was N601 Nanorouter USB and N740 Nanosensor from Sensinode [105]. An image of N601 and N740

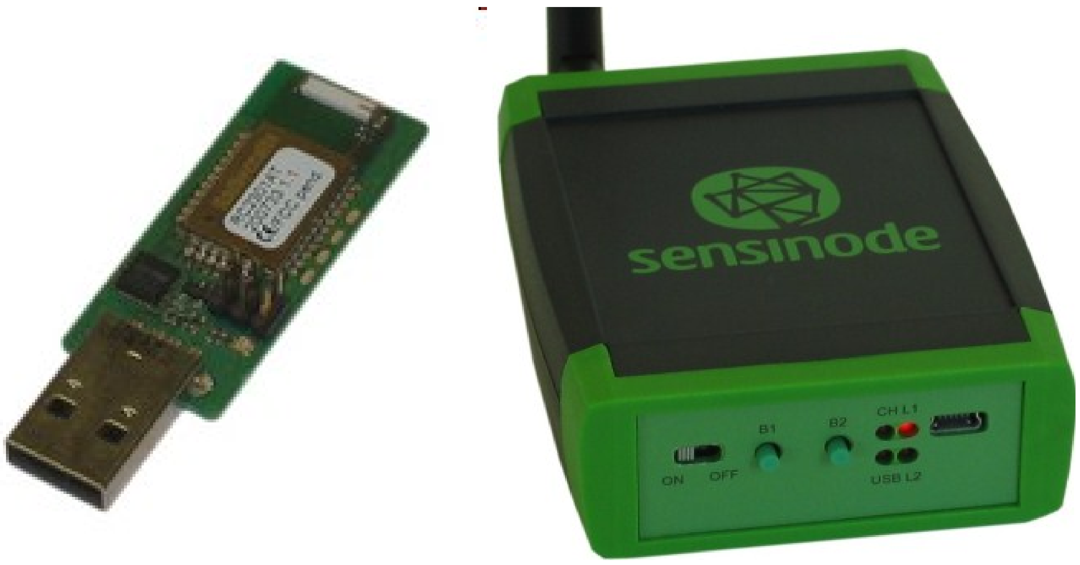

Figure 3.8: N601 Nanorouter USB and N740 Nanosensor from Sensinode

sensinode devices can be seen in Figure 3.8. These devices are equipped with the Texas Instruments cc2431 System-on-Chip *(SoC)*. The CC2431 System-on-Chip *(SoC)* solution is specifically tailored for IEEE 802.15.4 and ZigBee applications. In Sensinode devices, the CC2431 SoC has the following characteristics:

- CC2431 consists of a CC2430 SoC plus a location engine.
- Intel 8051 mcu *(CC2430 consists of a CC2420 and a 8051 mcu)*
- 2.4 GHz IEEE 802.15.4 compliant RF transceiver (cc2420)
- 128 KB flash, 8KB RAM

The CC2430 is highly suited for systems where ultra low power consumption is required. This is ensured by various operating modes. Short transition times between operating modes further ensure low power consumption [106].

3.4.1 8051 CPU and it's Memory

This section is very important because it explains how Sensinode devices operate and how the memory in these devices work.

CC2430 includes an 8-bit CPU core which is an enhanced version of the industry standard 8051 core. The enhanced 8051 execute instructions faster than the standard 8051 due to:

- The use of only one clock cycle per instruction *(CPI)* cycle compared to the 12 CPI used by the standard 8051.
- Elimination of wasted bus states

- Instruction cycles are aligned with memory fetch when possible and thus single byte instructions are performed in a single clock cycle.

8051 CPU architecture has four different memory spaces. The 8051 has separate memory spaces for program memory and data memory. Even though the four memory spaces are distinct in the 8051 architecture, in CC2430 are partially overlapping in order to ease DMA transfers and hardware debugger operation.

The Physical memory in CC2430 and the information stored in each part of the physical memory are:

- Flash: Program code and const data.
- Static RAM - *(S)RAM*: Data memory.
- Special Function Registers *(SFR)*: Hardware control.
- Flash Information Page *(Info Page)*: Device information and configuration.
- XREG: Additional registers.

The CC2430 memory spaces and an explanation of their mapping to physical ram is described in Table 3.1, while a detailed explanation of them follows [107]:

- CODE: Read-only program memory. Can address 64KB. This memory space maps the flash.
- DATA: Fast access (single instruction), read/write data memory. Addresses 256 Bytes. Maps the SRAM.
  - The lower 128 bytes of DATA can be addressed either directly or indirectly.
  - The upper 128 bytes of DATA can be addressed only indirectly.
- SFR: Read/Write register memory, directly accessible by a single CPU instruction. Addresses 128 bytes.
  - SFR registers whose address is divisible by eight are also bit-addressable.
  - XREGs are NOT mapped in SFR *(which is why they are not called SFRs)*.
- XDATA: Slow access *(usually 4-5 instruction cycles)*, 16-bit wide, read/write data memory. XDATA addresses the entire RAM (*including the parts addressed by DATA)*. It also addresses SFRs, parts of the flash, RF registers, XREGs. On the cc2530, XDATA also maps the Info Page.

Table 3.1: CC2430 Mapping between physical memory and hardware memory spaces

8051 Memory Spaces	Physical Memory
XDATA	SRAM, SFR, RFR, XREG, Flash, Info. Page
DATA	SRAM
CODE	Flash
SFR	SFR

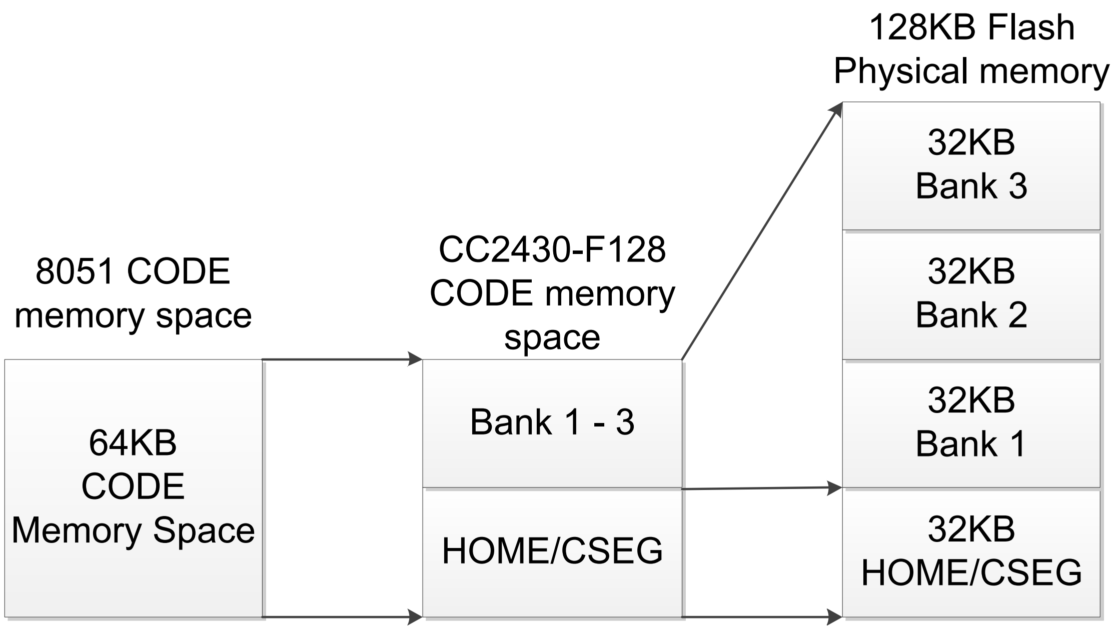

Figure 3.9: CC2430 CODE memory space

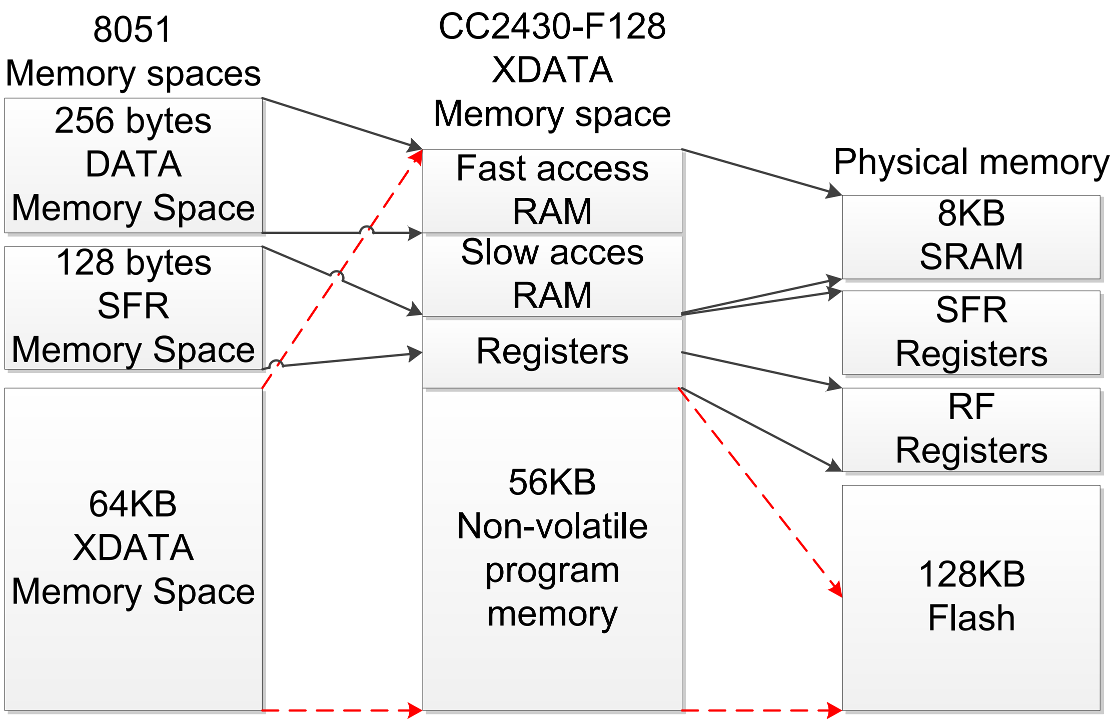

Figure 3.10: CC2430 XDATA memory space

Figure 3.9 and Figure 3.10 detailed illustrate how CODE and XDATA memory spaces are mapped in the physical memory.

In particular, DATA memory space is very important for the understanding of crucial modifications made in Contiki during this work. To be more precise, DATA hosts bit variables, R0-R7 register banks and variables placed there by the developer. What remains of DATA memory space will be the stack. Thus, the absolute theoretical maximum stack depth is 256 bytes, assuming nothing else resides in DATA. Contiki's 8051-based ports leave 223 bytes for the stack. Based on the above it is easy to conclude that a protocol developer that uses Contiki and 8051 based ports such as the CC2431 has very limited amount of stack and it is very likely to experience node crashes or even non functional nodes. More details about stack optimisations and necessary changes to Contikis codes for the functionality of the proposed mechanisms in this thesis can be found in Appendix D.

3.5 IEEE 802.15.4

IEEE 802.15.4 is a standard which specifies the physical layer and media access control for low-rate wireless personal area networks (LR-WPANs). The protocol stack can be seen in Figure 3.11. 802.15.4 intends to offer the fundamental lower network layers of a type of wireless personal area network *(WPAN)* which focuses on low-cost, low-speed ubiquitous communication between devices *(in contrast*

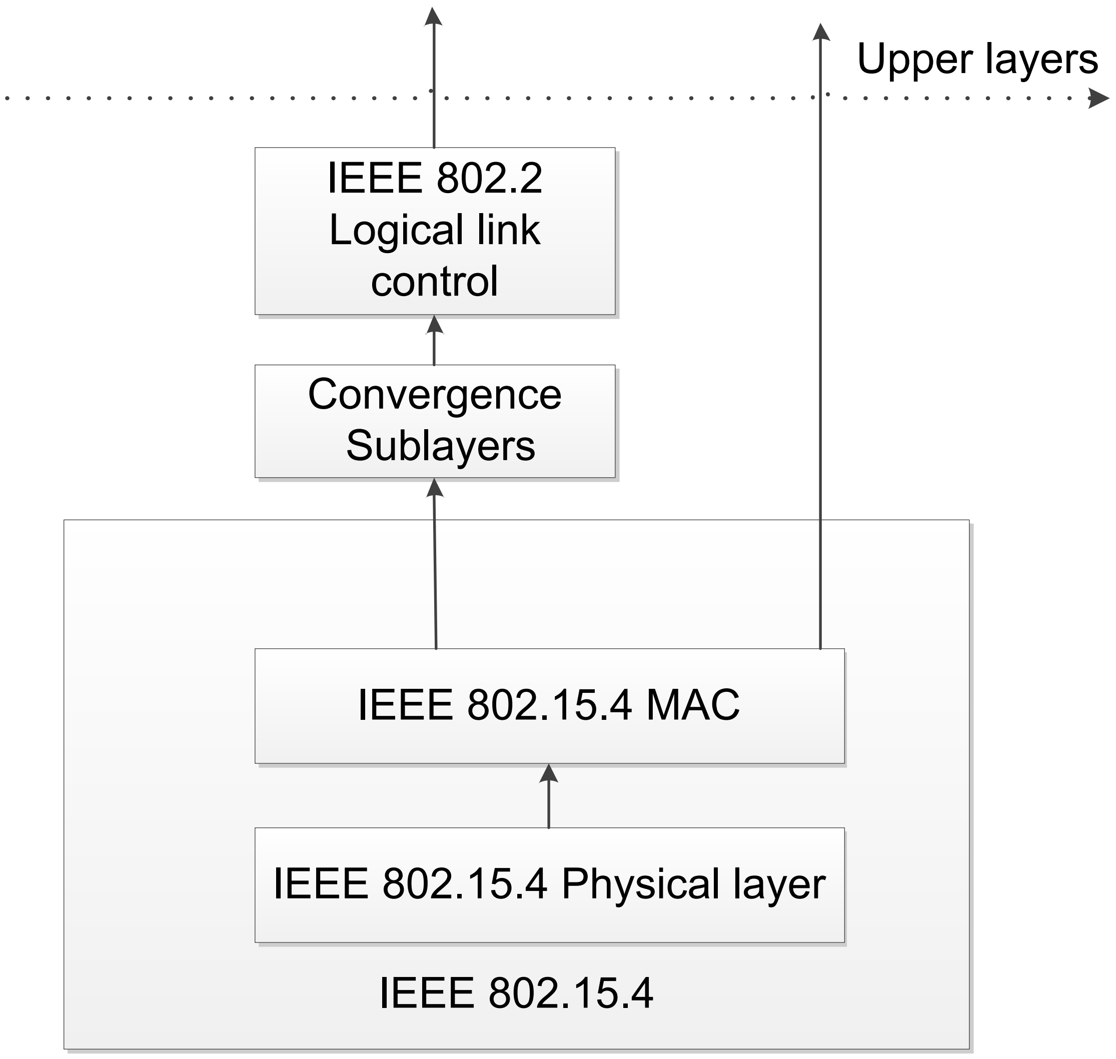

Figure 3.11: IEEE 802.15.4 protocol stack [3]

with other, more end user-oriented approaches, such as Wi-Fi). The emphasis is on very low cost communication of nearby devices with little or no underlying infrastructure, so that an even lower power consumption can be achieved [18, 19].

IEEE 802.15.4 networks are divided into PANs. Each PAN has a PAN coordinator and a set of PAN members. Packets sent over a PAN carry a 16-bit PAN identifier that specifies to what PAN the packet is destined. A device can participate in a PAN as a coordinator and simultaneously be a member in another PAN.

Each 802.15.4 node has a 64-bit address that uniquely identifies the device *(OUI: organisation unique identifiers).* 802.15.4 networks have limited packets, therefore the length of 64-bit addresses is prohibitive. For this reason, 802.15.4 allows nodes to use short addresses of 16 bits long. Long addresses are globally unique and each 802.15.4 device is assigned an address when manufactured. Short addresses are assigned at runtime by the PAN coordinator. A short address is valid only within the PAN in which it was assigned. It is also possible that devices can communicate with devices outside the PAN by using short addresses. This can be achieved by including 16-bit PAN identifiers, of both its own and the receiver devices PAN in the message.

The basic framework conceives a 10-meter communications area with a transfer rate of 250 kbit/s. Through the definition of multiple physical layers, embedded devices with lower power requirements can be benefited. Initially, the 802.15.4 transfer rates were defined between 20 and 40 kbit/s. In the current revision, another 100 kbit/s of rate has been added [3, 108].

Even lower rates can be considered with the resulting effect on power consumption. As already mentioned, the main identifying feature of 802.15.4 among WPAN's is the importance of achieving extremely low manufacturing and operation costs and technological simplicity, without sacrificing flexibility or generality.

Important features include real-time suitability by reservation of guaranteed time slots, collision avoidance through CSMA/CA and integrated support for secure communications. Devices also include power management functions such as link quality *(LQI)* and energy detection.

Although the definition of the network layers is based on the OSI model; only the lower layers are defined in the standard. In order to interact with upper layers an IEEE 802.2 logical link control sublayer that accesses the MAC through a convergence sublayer must be used [20, 21].

3.6 Radio Duty Cycle

In general, it is believed that packet transmissions *(TX)* consume more energy than packet reception *(RX)* or radio idle listen *(lots of the previous works, only measure TX and sometimes RX for the calculation of energy consumption)*. This assumption is incorrect since radio RX and idle listening consume as much energy as radio TX. Therefore, if the sensor network is not configured with a duty cycle algorithm, the energy consumption of idle nodes will be similar to the one of the active nodes.

In order to justify the above statements, I further investigated the energy consumption of Sensinode *(CC2430 SoC; used for the test-bed experiments)* and Tmotesky *(CC2420 Radio transceiver; used for the simulation experiments)* nodes. The former architecture [106], has an electrical current of:

Radio RX/Idle listen: 19.2 mA

Radio TX: 19.4 mA

and average electrical potential difference *(Voltage)* of 3. This in turn means that the power consumption for Sensinode will be:

Radio RX/Idle listen: 0.0576 Watt

Radio TX: 0.0582 Watt

therefore the energy consumption of Sensinode is:

Radio RX/Idle listen: 0.0576 Joule for every second in this state.

Radio TX: 0.0582 Joule for every second in this state.

The later architecture [109], has an electrical current of:

Radio RX/Idle listen: 19.7 mA

Radio TX: 17.4 mA

and average electrical potential difference *(Voltage)* of 2.85. This in turn means that the power consumption for Tmotesky will be:

radio RX/Idle listen: 0.056145 Watt

radio TX: 0.04959 Watt

therefore the energy consumption of Tmotesky is:

radio RX/Idle listen: 0.056145 Joule for every second in this state.

radio TX: 0.04959 Joule for every second in this state.

According to the above calculations, it is clear that the energy consumption is similar for the radio states of TX and RX/idle-listen. In some cases such as the Tmotesky nodes, radio RX/idle-listen had noticeably higher energy consumption. Consequently, a sensor network may have a significant waste of energy during idle operation. During the last years, in order to prolong the networks life-time, significant efforts has been made for the development of duty cycling MAC protocols. With radio duty cycling, nodes will turn off their transceivers for long periods of time in order to minimise idle listening [6]. The majority of the existing DC MACs use a synchronisation algorithm, in order to turn on their radios almost simultaneously, exchange data and go back to sleep mode. A smaller part of DC MACs such as ContikiMAC, are using continuous packet transmissions in order achieve communication between nodes without the operation of a node synchronisation algorithm. These protocols will be described in detail in subsection 3.6.1.

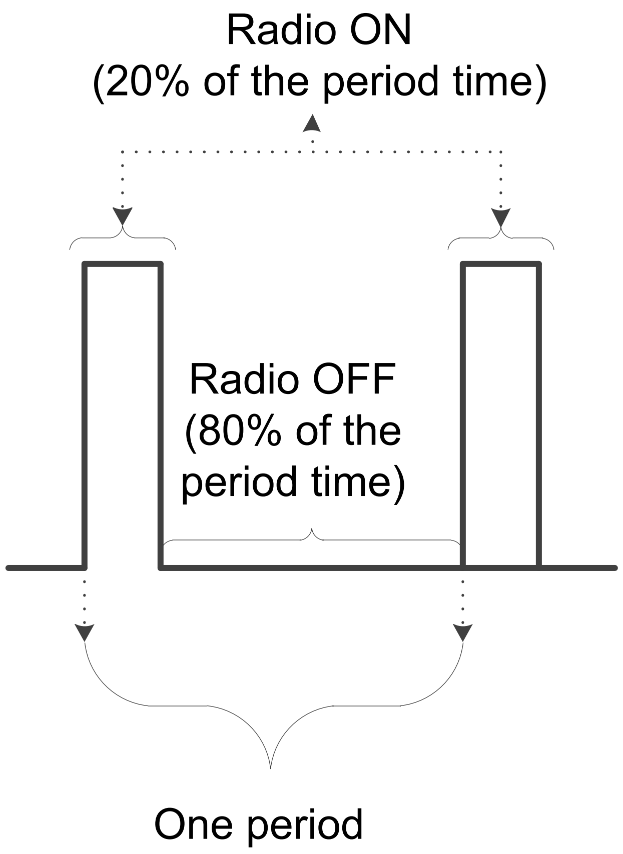

Figure 3.12: an example of a 20% duty cycle

In literature, duty cycle is the time that an entity spends in an active state as a fraction of the total time under consideration [110, 111, 112].

In sensor nodes, a 20% duty cycle means the radio transceiver is on for 20% and off for 80% of the time. The *on time* for a 20% duty cycle could be a fraction of mseconds, seconds, or even days depending on how long the device's period is. Hence one period is the length of time it takes for the device to go through a complete on/off cycle [113]. Figure 3.12 demonstrates 20% duty cycle.

In a periodic event, the duty cycle is the ratio of the duration of the event to the total period of a signal [114]:

$$D = \frac{\tau}{T} \tag{3.1}$$

were:

D is the duty cycle.

τ is the duration that the function is active.

T is the period of the function.

Duty cycle algorithms can be divided in two categories:

Asynchronous duty cycle achieves low-power operation by switching the radio off most of the time and periodically switching it on for a short while. By keeping the radio on for a short period, it is possible for nodes to detect and receive traffic from neighbour nodes. Protocols tailored based on this approach, need to send a train of strobes or packets. When the receiver hears an incoming strobe or packet it keeps the radio on for the full packet reception. Asynchronous duty cycle protocols are more simple and implicitly synchronise themselves on every data transmission.

Synchronous duty cycle protocols are built on time synchronisation by explicitly synchronising themselves before sending any data packets *can be with the use of time synchronisation mesh protocols, WSNIP*. Several methods for time synchronisation exist [115, 116].

3.6.1 Radio Duty Cycle in Contiki

In Contiki, duty cycle is performed by the Radio Duty Cycling (*RDC)* layer. In Contiki, the radio duty cycle is expressed as a function of the wake up frequency called channel check rate *(a channel check rate of 8 will result in 8 wake-ups a second for each node)*. Contiki provides a set of RDC mechanisms, with various properties such as: X-MAC, LPP, and ContikiMAC. All three duty cycle protocols in Contiki are asynchronous. The default mechanism in Contiki is ContikiMAC [117, 118].

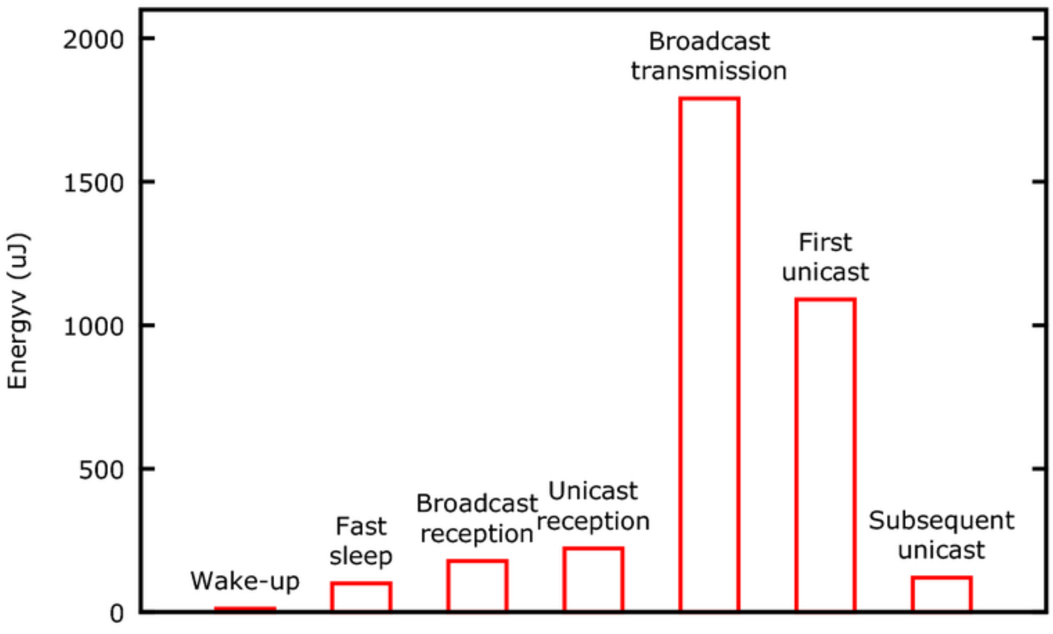

Figure 3.13: Energy consumption of the individual ContikiMAC operations [4]

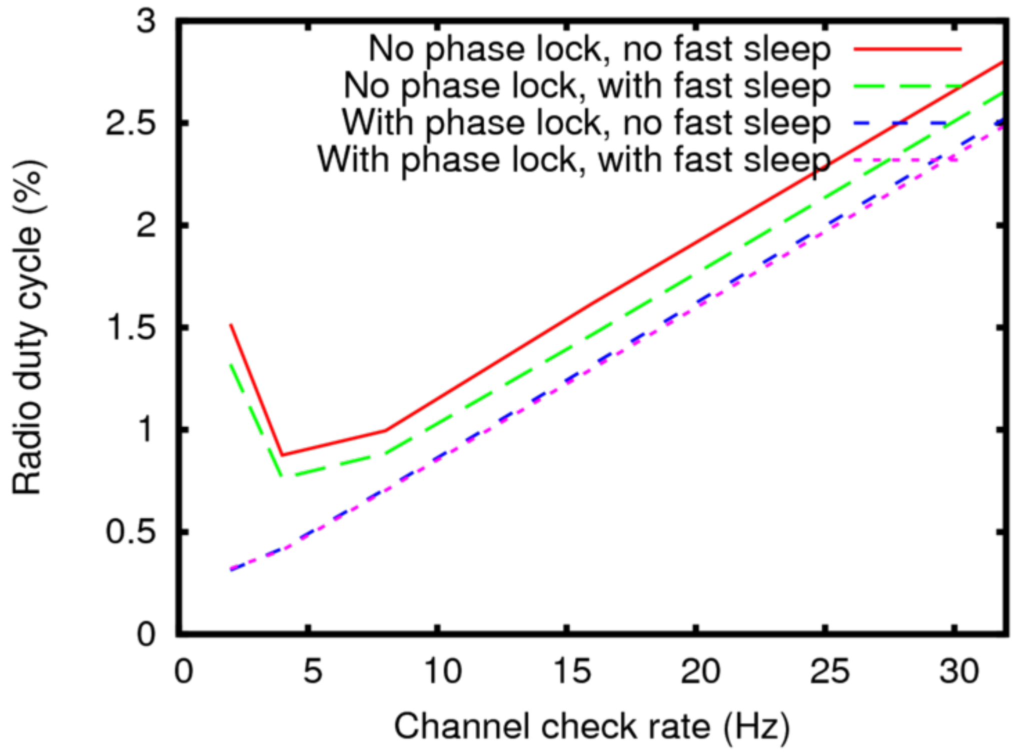

Figure 3.14: The network radio duty cycle with ContikiMAC, averaged for all nodes ta the network without path loss [4]

ContikiMAC [4] uses periodical wake-ups in order to listen for packet transmissions from neighbour nodes. If a packet transmission is detected during a wake-up, the receiver keeps the radio on to receive the packet. When a packet is received, the receiver sends a radio acknowledgment. In each packet transmission, sender is repeatedly sending the packet until an acknowledgment is received. Broadcast packets doe not wait for link layer acknowledgments. Instead the transmitter node will continuously send the packet for the whole wake-up interval. Additionally, ContikiMAC uses a fast sleep optimisation, to allow receivers to quickly detect false-positive wake-ups, and a transmission phase-lock optimisation, to allow runtime optimisation of the energy-efficiency of transmissions. Furthermore, a power-efficient wake-up mechanism that relies on precise timing between transmissions is implemented in ContikiMAC. An inexpensive Clear Channel Assessment (CCA) mechanism that uses the Received Signal Strentgh Indicator (RSSI) of the radio transceiver to give an indication of radio activity on the channel is used for ContikiMAC's wake-up. If the RSSI is below a given threshold, the CCA returns positive, indicating that the channel is clear and thus node returns to sleep mode. If CCA returns negative, indicates that the channel is in use. Therefore the radio remains on for the reception of the transmitted frame. Figure 3.13 shows the energy consumption for the individual ContikiMAC operations. It is visible that wake-up

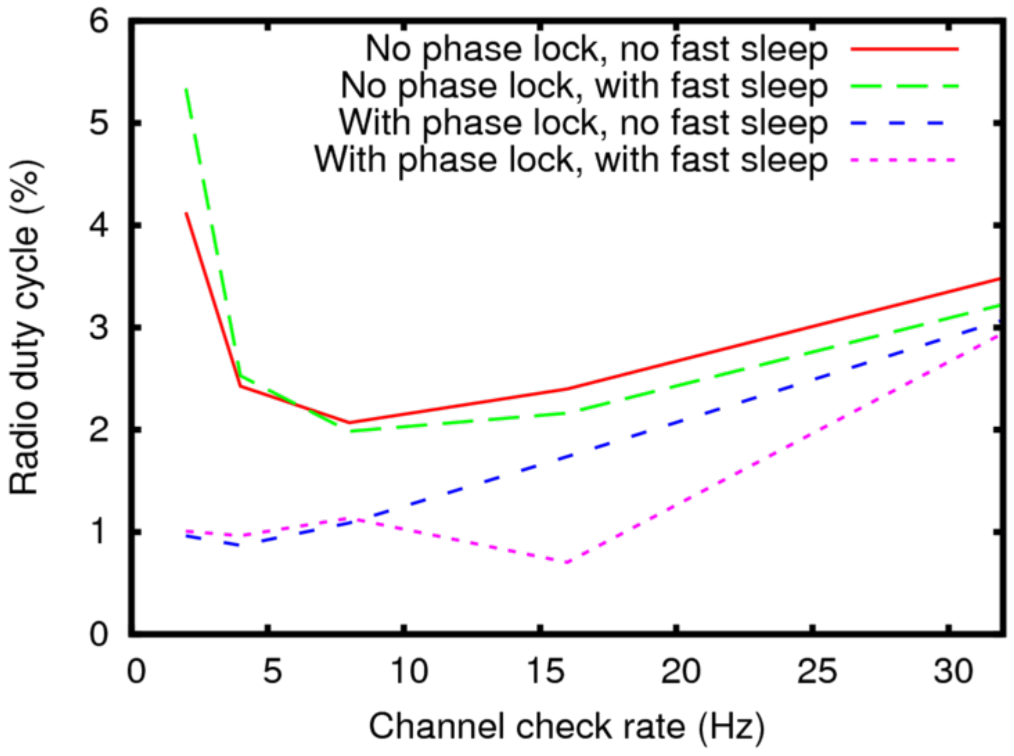

Figure 3.15: The network radio duty cycle with ContikiMAC, averaged for all nodes in a network with path loss [4]

has the lowest energy cost. This can explain why ContikiMAC is designed to operate with frequent *(many times per second)* wake-ups and CCA checks. Figure 3.14 and Figure 3.15 demonstrate the radio duty cycle in ContikiMAC, averaged for all nodes in the network for no-loss and with loss paths accordingly.

In X-MAC [8], before each packet transmission, nodes transmit short preambles that contain the destination address. Periodically, a node will switch its radio on and scan for incoming preambles. When a node is not included in a communication *(no preamble received or preamble is not destined for that node)*, it will immediately go back to sleep mode. When a node successfully receives a preamble, it will reply with an early acknowledgment (ACK) and keep the radio listening in order to receive the incoming packet. X-MAC assume that nodes will wake-up simultaneously after a fixed interval. Figure 3.16, demonstrates the radio duty cycle in a data collection network with path loss for X-MAC and ContikiMAC.

R. Musaloiu-E et al. propose Koala: an Ultra-Low Power Data Retrieval in Wireless Sensor Networks [9]. In this work a low power probing *(LPP)* technique has been proposed for duty cycling. Nodes will periodically broadcast short packets *(probes)* requesting acknowledgments. When a node receives an acknowledgment, it remains active and starts waking up other nodes by acknowledging their probes. If a node does not receive an acknowledgment after transmitting the

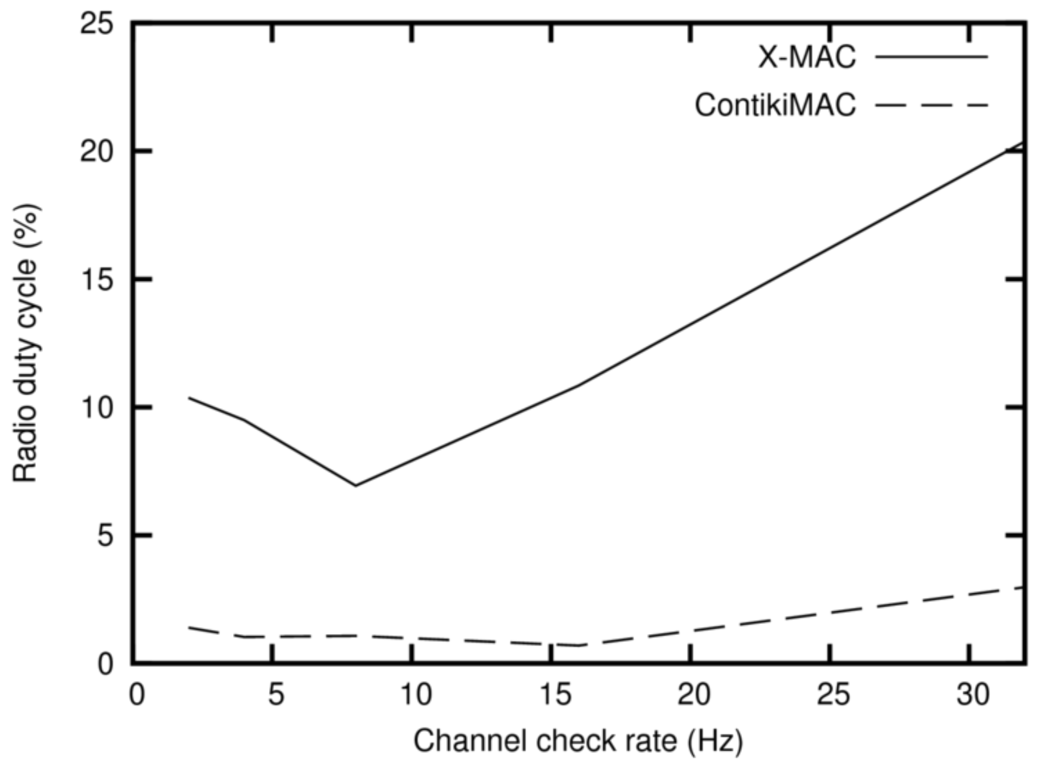

Figure 3.16: The radio duty cycle in a data collection network with path loss, with X-MAC and ContikiMAC, as a function of the wake up frequency called channel check rate [4]

probe packets, it will go immediately back to sleep.

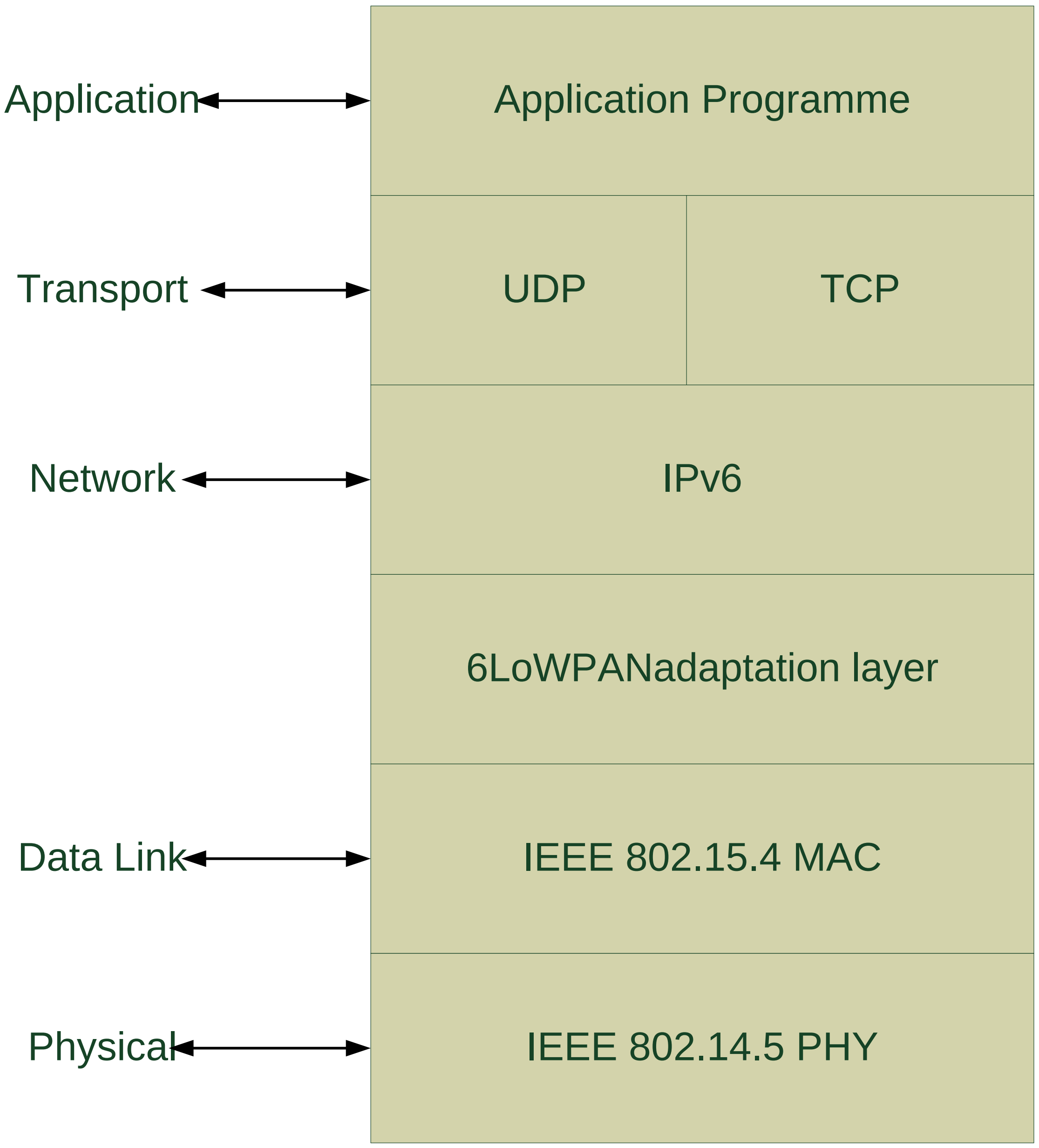

Figure 3.17: 6LoWPAN adaptation layer

3.7 6LoWPAN

In subsection 2.2.1 we discussed what are the advantages of integrating IP in sensor nodes. However, to integrate IP in WSNs, several significant attributes must be combined. WSNs are data centric while IP networks are address centric. The main objective of 6lowPAN, proposed by IETF, is to integrate IPv6 in LoWPANs supported by IEEE 802.15.4 [119, 120, 121, 122, 123].

IPv6's MTU is 1280 bytes. IEEE 802.15.4 standard defines a packet size of 127 bytes. Out of the 127 bytes of 802.15.4, 25 are used by the MAC layer headers and optionally 21 bytes are consumed for security by AES-CCM-128. In the worst case this leaves 81 bytes for the IPv6 payload. After removing the size of an IPv6

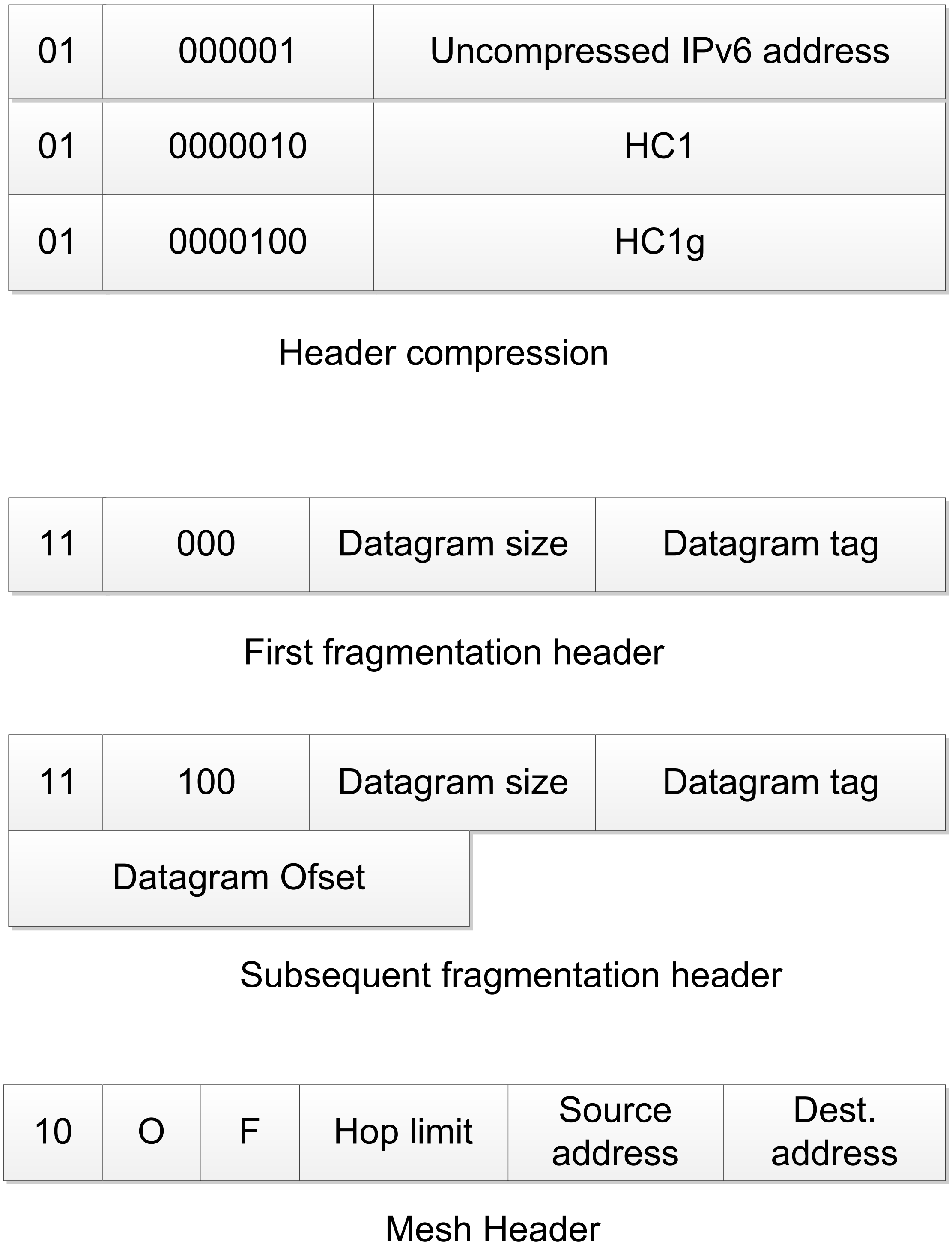

Figure 3.18: Layout of 6LoWPAN headers

header *(40 bytes)* only 41 bytes are left. Additionally, the transport layer header must be deducted from the remaining 41 bytes *(8byte UDP header and 20 bytes the TCP header)*. This would lead to a very short payload *(33 bytes if UDP is used and 21 bytes if TCP is used)*.

Based on the above, an adaptation layer is needed to comply with the IPv6 requirement to support a minimum MTU size of 1280 bytes as well as compression techniques to reduce protocol overhead. RFCs 4919 [22] and 4944 [23], define the functions included in 6LoWPAN. The 6LoWPAN adaptation layer provides three main services:

- Packet size adaptation, fragmentation and reassembly in order to fragment IPv6's packets into 127 byte packets.
- Header compression. This feature allows the protocol to compress the 40 bytes of standard IPV6 to just 2 bytes.
- Link layer *(layer 2)* forwarding when multi-hop is used by the link layer.

In most cases, the use of efficient compression allows most applications to send their data within a single IPv6 packet. Figure 3.17 demonstrates an IPv6 with 6LoWPAN protocol stack.

Similar to IPv6, the 6LoWPAN adaptation layer makes use of header stacking *(headers are added only when needed)*. Currently three type of headers are supported by 6LoWPAN:

- A mesh addressing header.
- A fragment header.
- An IPv6 compression header.

These headers will appear in the above order when present. Figure 3.18 shows the layout of 6LoWPAN headers.

3.8 uIP other IP Stacks for Embedded Systems

For many years, it was believed that IP was too complex and heavyweight to be usable in sensor nodes since the microcontrollers used by the sensors are constrained in memory size and processing power. In general, an IP stack *(in Linux)* requires at least one megabyte of memory. In contrast, sensor nodes typically have a few kilobytes of memory [5, 124, 125].

uIP stack is an implementation of the IP stack specifically designed to meet the memory requirements of sensor nodes and other embedded systems [126, 127].

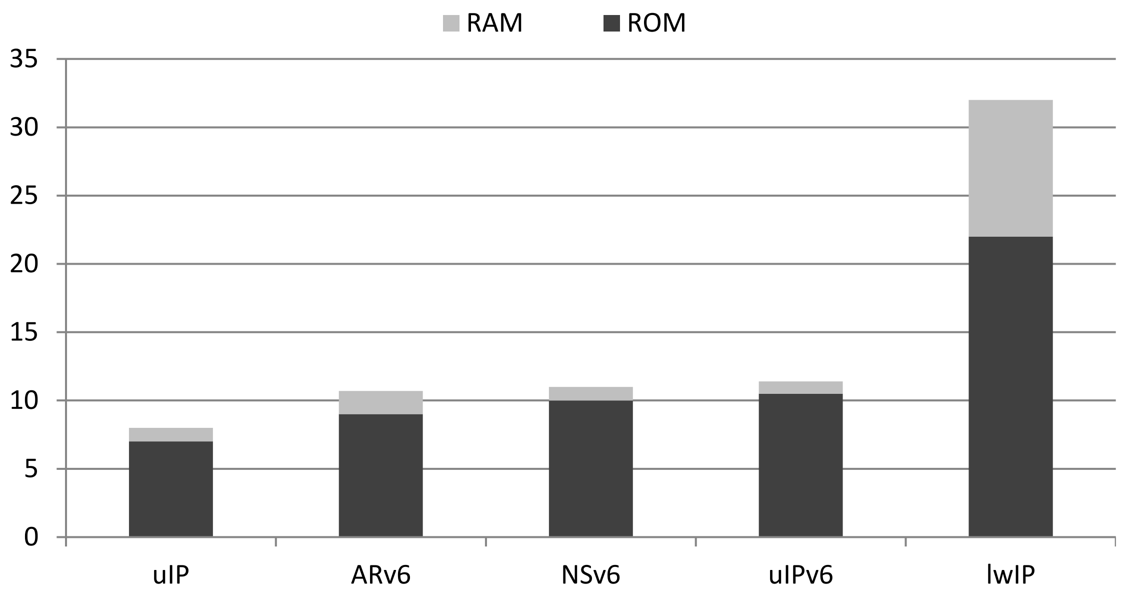

Figure 3.19: The memory footprint for uIP and other commercialy available IPv6 stacks [5]

uIP was first released in September 2001 under permissive open source license that allows the software to be used freely in commercial and non-commercial systems. Since its release, uIP has seen a significant adoption.

uIP has very low memory requirements of 1KB of RAM and a few KB of ROM in its initial configuration. The initial configuration includes IP, ICMP, UDP and TCP protocols. More specifically, uIP's code size depends on the processor in which the stack will be used. It is possible to further reduce the RAM requirements but at expense of standard compliance. A uIP configuration can be as small as 100 bytes of RAM.

lwIP stack [126] is another implementation of IP for embedded systems. LwIP is designed for slightly larger systems than the uIP stack and thus it has larger memory requirements. A typical installation of lwIP stack requires 40KB of RAM and 20KB of ROM. Even though lwIP has higher memory requirements than uIP it can achieve higher performance.

uIP stack implements the network and transport layer protocols of the IP protocol family such as: IP, ICMP, UDP and TCP. Additionally, uIP was the first IP protocol stack for embedded systems to implement a fully compatible with the standards TCP protocol. In 2008, Cisco Systems extended uIP with IPv6 capabilities. UIPv6 was the first stack to comply with all IPv6 requirements [128, 129].

Originaly, uIP was designed to be used either with or without an operating system. Today many operating systems use uIP for IP communications. Contiki OS uses uIP as its primary IP communication stack. FreeRTOS provides a choice between uIP and lwIP while TinyOS uses uIP for IPv4 communications. Figure 3.19 demonstrates a memory requirements comparison between the existing commercial IP stacks for embedded systems.

The three main methods used by uIP for the reduction of code size are:

- an event-driven programming interface
- a simple buffer management scheme
- a memory efficient TCP implementation.

3.9 RPL: Routing Over Low Power and Lossy Networks

As discussed in subsection 2.2.3, in 2008, the Internet Engineering Task Force *(IETF)* formed a new group called ROLL *(Routing Over Low-Power and Lossy networks)* in order to produce a set of routing requirements and determine whether or not existing IETF routing protocols can satisfy these requirements. The working group quickly converted on the fact that none of the existing routing protocols would satisfy the fairly unique set of routing requirements for LLN. This led to the design of the new emerging standard for routing in LLN named Routing for Low Power and Lossy Networks *(RPL)*. RPL [130, 131] is still a work in progress and the IETF RFC [132] should be used as reference. Various aspects may change or be added to the specification.

A lossy link is a link with significantly higher Bit Error Rates *(BER)* than traditional Ethernet and optical links. Packet losses in lossy links are extremely frequent, and links may even become totally unusable for quite some time for multiple reasons such as interference. This observation is one of the most important factors in the design of protocols for lossy links. Knowing that link failures are frequent and usually transient means that a routing protocol should not overreact to network instabilities or try to stabilise under unstable conditions. Routing protocols designed for non lossy links can lead to routing instabilities and generate significant amount of control traffic which is costly for the whole network [5].

A routing protocol designed for LLN must be able to determine whether or not a link should be considered as down and consequently inadequate for traffic forwarding. The same reasoning applies for the decision of whether the link is usable or not. When a link is used for a communication, it must be observed carefully in order to determine if it remains usable or not*(link state may change in LLN)*. Furthermore, since the resources are scarce the control traffic must be tightly bounded for bandwidth and energy saving.

As specified in [133, 134], RPL is a distance vector protocol that builds Destination Oriented Directed Acyclic Graph *(DODAG)* where paths are constructed from each node in the network towards the DODAG root *(usually a sink or border router node)*. There are numerous reasons behind the distance vector design of RPL. The main reason is that link state routing protocols are more powerful and thus they require a much greater amount of resources such as memory*(routing tables)* and control traffic *(synchronise the link state databases)*.

A DODAG is a set of vertices connected by directed edges with no directed cycles. For each DODAG, RPL is forming a set of paths from each leaf node towards the DODAG root. The routing paths within a DODAG are redundant which is an important requirement for LLN. Therefore if the topology permits, RPL may provide multiple paths between the leaf nodes and the DODAG root. In each DODAG, one or more nodes can be configured as the root by the administrator. The node discovery mechanism in RPL use the newly defined ICMPv6 messages. RPL defines two new ICMPv6 messages called DODAG information object *(DIO)* and destination advertisement object *(DAO)*.

DIO messages are sent by nodes in order to advertise information about the DODAG, along with other DODAG parameters such as path metrics. When a node discovers multiple DODAG neighbours, it makes use of various rules to decide whether to join the DODAG *(currently in Contiki each node can participate in only one DODAG)*. When a node joins a DODAG, it has a route towards the DODAG root *(up traffic; leaf nodes to sink)*.

In order to provide routing information in the down direction *(sink to leaf nodes)*, RPL uses the DAO messages. DAO messages are used to simply advertise prefix reachability toward the leaf nodes. DAO messages, carry prefix information as well as the lifetime of the message and the depth of the path or cost information to determine how far the destination is.

Another type of message used by RPL is the DODAG information solicitation message *(DIS)*. DIS messages are similar to the IPv6 router solicitation *(RS)* message; used to discover DODAGs in the neighbourhood and solicit DIO messages from the RPL nodes in that neighbourhood. DIS messages have no additional body *(information)*.

The transmission of DIO and DAO messages in RPL is governed by the use of trickle timers [135]. With Trickle, dynamic timers that govern the sending of RPL control messages and attempt to reduce redundant messages are used *as discussed in the previous paragraphs*. When a DODAG is unstable, RPL messages are sent more frequently. On the contrary, as a DODAG stabilises RPL messages become less frequent.

When RPL links fail, paths are repaired using the local and global repair mechanisms. The former, quickly finds a new backup path without an attempt to globally optimise the whole DODAG. The later, rely on re-optimisation process driven by the DODAG root.

In RPL, each node has a node rank which is determined through the objective function *(OF)*. RPL can be configured to operate with two OF:

1. Link quality level *(LQL)* is used as the global recorded metric and favors paths with the minimum of low and fair quality links.

2. Distance between the leaf and the RPL root nodes.

Routing loops are always undesirable and one of the objectives of routing protocols is to avoid the formation of loops whenever possible. RPL, does not guarantee the absence of temporary loops; instead it tries to avoid loops by using loop detection mechanism via data path validation. In order to avoid loops, RPL follows two main rules:

1. A node is not allowed to select as a parent a node with higher rank than the node's rank+DAGMaxRankIncrease.

2. A node is not allowed to be greedy and attempt to move deeper in the DODAG in order to increase the selection of DODAG parents.

Even with the two loop avoidance mechanisms stated earlier, loops may take place in a number of circumstances such as lost DIO messages and failed attempts to inform parents about lost paths through DAO messages. Potential ways to solve this problems include the acknowledgment of DAO messages. Since loops are hardly avoidable, loop detection mechanisms must be available. The loop detection mechanism in RPL, piggybacks RPL data in the data packets by setting flags in the packet header *(the exact location where these flags are carried is not yet defined)*. The idea behind the flags is to verify that the packets are making forward progress in order to detect loops or DODAG inconsistencies [5].

3.9.1 RPL What Went Wrong?

In recent studies T. Clausen et al. has presented a critical evaluation of RPL routing protocol [136]. This evaluation provides an insight of the limits and weaknesses of RPL.

The evaluation in [136], shows that DIO message generation/processing rules and the trickle timers are straight forward and the state required at each RPL router is minimal. On the contrary, the mechanism for DAO messages is less elegant and thus problems include underspecifications such as:

1. If a DAO message is not sent before the time of the previously sent DAO expires, the routing entry will not be renewed and thus there is a high risk of data traffic loss.

2. RPL does not specify any jitter between packet transmissions. Therefore if DAOs sent periodically, adjacent routes may transmit DAO messages at the same time which in turn lead to link layer collisions.

3. Non-storing mode calculate routes based on a "piece-wise calculation,". This approach relies on previous reception of DAOs from intermediate routes along the path. Consequently, if some of these DAOs are not received; route calculations is impossible and thus data traffic cannot be sent to the destination.

Additionally to the above listed underspecifications, RPL suffers from rooting loops. Authors in [136] has experimentally demonstrated that routing loops can occur with the current implementations of RPL. Routing loops can occur in both strong and non-storing mode. Even though loops can be detected during the construction of the source-route, the only corrective measure that the DODAG root can take is to trigger global repair and thus a complete rooting reboot in the LLN.

In Contiki, only strong mode is implemented. Therefore the above observations can occur during the experiments. To avoid this the network administrator should always be aware of the routing state in the network. How this can be achieved will be explained in section 3.10

3.10 Setting Up a Test-bed

The majority of experimental works on WSNs are based on results from simulators. Network simulators may not always be reliable [137]. Furthermore, most network simulators assume that the environment is ideal and only some of them provide features for the simulation of real environmental parameters during the simulations. This has led to the conclusion that real test-bed experiments are of great importance for the evaluation of protocols under real environment. A test-bed can be defined as:

"A controlled environment for experimentation and evaluation, with metrics and benchmark content that allow comparison of tools and strategies"

A test-beds, can be deployed both indoors and outdoors. There are numerous parameters that play an important role for this decision. These parameters can be the design of the nodes *(e.g waterproof casing at the nodes)*, the season of the year, the security in the area the sensors will be deployed *(nodes can be stolen)*, infrastructure in the test-bed area *(power supply for laptops or nodes etcetera)* or even the duration of the experiment/measurements. Usually outdoor test-beds demonstrate more stable behavior due to the reduced interference *(people, electric machines, wifi etcetera)*. On the other hand, it is easier and more secure to deploy an indoors testbed. In this work, due to the characteristics of the devices and the environmental conditions the test-beds deployed for the evaluation of the algorithms were indoors. The rest of this section describes the steps and procedures required for the successful deployment of indoor test-beds *(The following procedures were used every time we deployed a testbed)*.

Step 1: Pick the optimal location. Analysing the area that the test will be deployed is very important since different dimension rooms, corridors, distance and objects *(video devices such as satellite TV, microwave ovens or even car alarms)* can result in decay of the received signal strength which in turn can cause link failures and routing path readjustments [138]. Therefore the placement of the sensor nodes must be considered carefully. This can be achieved by using some sensors to collect the LQI and RSSI values for each packet transmitted over the different links. Based on the collected values from the different node positions - distances; a decision for the placement of the sensor nodes and the deployment of the test-bed can be made.

Step 2: Reduce the environmental interference. As mentioned previously, the environment can dramatically affect the behaviour at each node. As a result eliminating as many environmental variables as possible, such as people interference can lead to more accurate measurements. Running the sensor experiments during low peak times *(not many people around)* can significantly help in attaining more consistent results between different experiment runs. To explain how this can affect a testbed lets assume two scenarios:

1. different positioning on the doors, which can potentially lead to the creation of different routing paths *(due to the changed RSSI/LQI and thus depending on the OF of the routing protocol, node ranks can be changed)*. Combining frequent changes in the environment such as doors positioning with the a purposely unstable routing protocol such as RPL, can lead to confusing results *(3 hops away node may re-pick a parent and end up 2 hops away)* and thus significant delay in the evaluation of algorithms or even discarding of hours of experiment results.

2. 802.11 Wifi access points are installed in every building nowadays. Wifi operates in the same radio frequency *(2.4 GHz)* as the 802.15.4 used by the sensors. When wifi activity is high, 802.15.4 performance can significantly be affected *(in some cases nodes can become totally unable to communicate with each other)*.

Step 3: Reduce the wifi interference. If the area that the test-bed will be deployed incorporates wifi access points, potential interference with the 802.15.4 sensors may lead to undesirable behaviours or even impossibility to conduct experiments. In such cases, it will be wise to configure the 802.15.4 sensors with a non overlapping channel. Figure 3.20 shows the 26 802.15.4 channels and which channels overlap with the 802.11. Therefore, 802.15.4 channels 25 and 26 are usually the most popular choice for the configuration of 802.15.4 devices. This in turn may lead to potential interference from other 802.15.4 devices in the area and thus similar problems may occur during the experiments. Based on the above, in order to configure the 802.15.4 devices with a low-interference channel, a complete knowledge of the status in every 802.15.4 channel is required. This knowledge can be achieved with the use of an energy scan. Energy scan is usually one device that periodically scans every 802.15.4 channel and then returns the energy detected from the radio in every channel *(this can be achieved by measuring the RSSI values)*. More details about the energy scan can be found in Appendix B. During our experiments, a sensor was configured as the energy scan in order to ensure that there wasn't any interference at the testbed from outside devices.

Step 4: What is the optimal data collection method? Collecting the desired data in a test-bed is not as easy and straightforward as it is in network simulators. Usually in a test-bed, the majority of nodes are not directly connected to devices such as PCs and thus collection of data from the sensors becomes harder. In order to collect data in WSN methods such as storing of the data in external flash or through network packets are used. When the data are stored in external flash, nodes have to be collected for the retrieval of the information *(or requested through the network)*. On the other hand, when nodes periodically transmit their data through network, the base station will receive and process the information. Since data collection constitutes one of the most significant parts of the test-bed and redeploying a test-bed is time-consuming and hard, the network architects must carefully consider what network data are required for their evaluation. Likewise, the sensor applications must be adjusted to successfully collect and deliver the data to the base station.

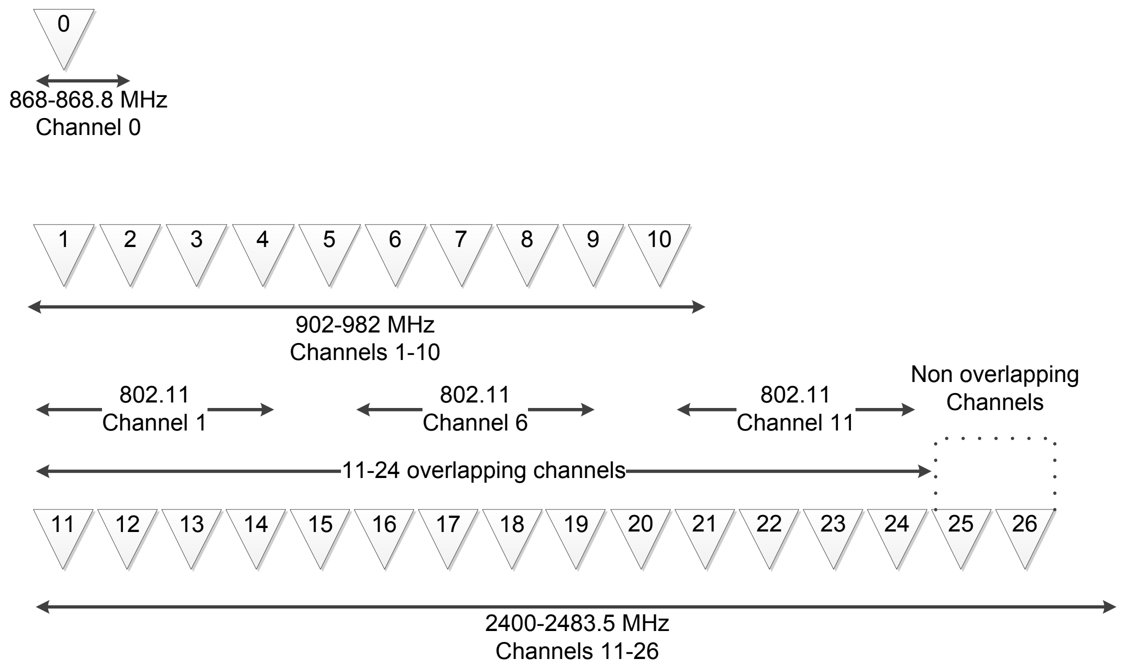

Figure 3.20: The 26 defined by the IEEE 802.15.4 channels and how they overlap with 802.11 channels

Step 5: Sett up the transmission range at the devices. The transmission range of 802.15.4 devices is usually up to 75 meters. This can also differ based on the antenna used by the devices. Based on the 802.15.4 transmission ranges, in order to deploy a multi-device and multi-hop network a huge area would be required. Therefore, the transmission range at each node should be configured based on the area where the test-bed will be deployed *(not always the default)*.

Step 6: Avoid frequent device reconfiguration. According to the experiments needs, environmental parameters such as the ones described in steps 3 - 5 and the protocols to be evaluated in the testbed; different images of the OS should be build for the sensor nodes. To explain this in detail, lets assume that there are 2 MAC protocols to be tested. This will require two different images of the operating system *(MAC protocols can also be switched on the fly but this will result in a larger image in terms of memory. Sensor nodes do not have large amounts of memory and thus this may not be possible in various cases)*. Configuring the nodes with different images usually means that every node must be reprogrammed. Reprogramming the nodes can be a very time consuming and hard procedure. For our testbed, the various Contiki OS images were saved in the external Flash. Each node was programmed with a boot-loader software *(software which allows the user to load different images when the node boots)*. More information about the bootloader can be found in Appendix B. With this approach, the OS image at the nodes can be switched every time the nodes reboot. In some cases, nodes do not have external flash memory and thus this approach is not possible. Consequently, other node reprogramming methods such as Over The Air Programming *(OTA)* should be used.

Step 7: Auto device programming. Programming the sensor nodes can be very time consuming, especially when there is a large number of sensor nodes. In order to speed up this procedure, we developed an auto programming script *it program all the devices connected to the USB interface with the desired OS image.* A script such as the above can significantly speed up the node programming process.

Step 8: Internally monitor the network. At least one node in the network must be the border router. The border router is the RPL route node and is used to route IPv6 traffic between the 6LoWPAN and the Internet. More specifically, in our experiments, 6LoWPAN traffic is forwarded by the border router to the tuneslip6 interface from which it then is received by a data collection sever. The data collection server used in our testbed will be described in detail at section 3.11. In addition, a network visualisation software is developed and included in Contiki OS. Through this software, the routing tree and routing information can be retrieved from the nodes at any time. A tool like this is very useful for the understanding of the sensor node behaviour and debugging of the network.

Step 9: Externally monitor the network. During step 3, it was mentioned that an energy scan should be used in order to decide which 802.15.4 channel is the most appropriate for use for the deployment of the test-bed. It is suggested that the energy scan is attached to a mobile device. This way scanning the status of the channels in various areas of the testbed can be achieved. In addition to the energy scan, it is wise to have a packet sniffer device. A packet sniffer is usually a sensor that can scan the radio *(it scans only one channel; usually the one that the test-bed devices are configured with)* and capture the packets transmitted in the area. With the use of packet sniffers, debugging the network and understanding the behaviour of the protocols becomes much easier. Through our experience, a packet sniffer should be used in every sensor network for debugging purposes. More details about the packet sniffer used for our experiments can be found in Appendix B.

Step 10: Set up the desired routing protocol. Configuring the network with a routing protocol is very important since the network will demonstrate

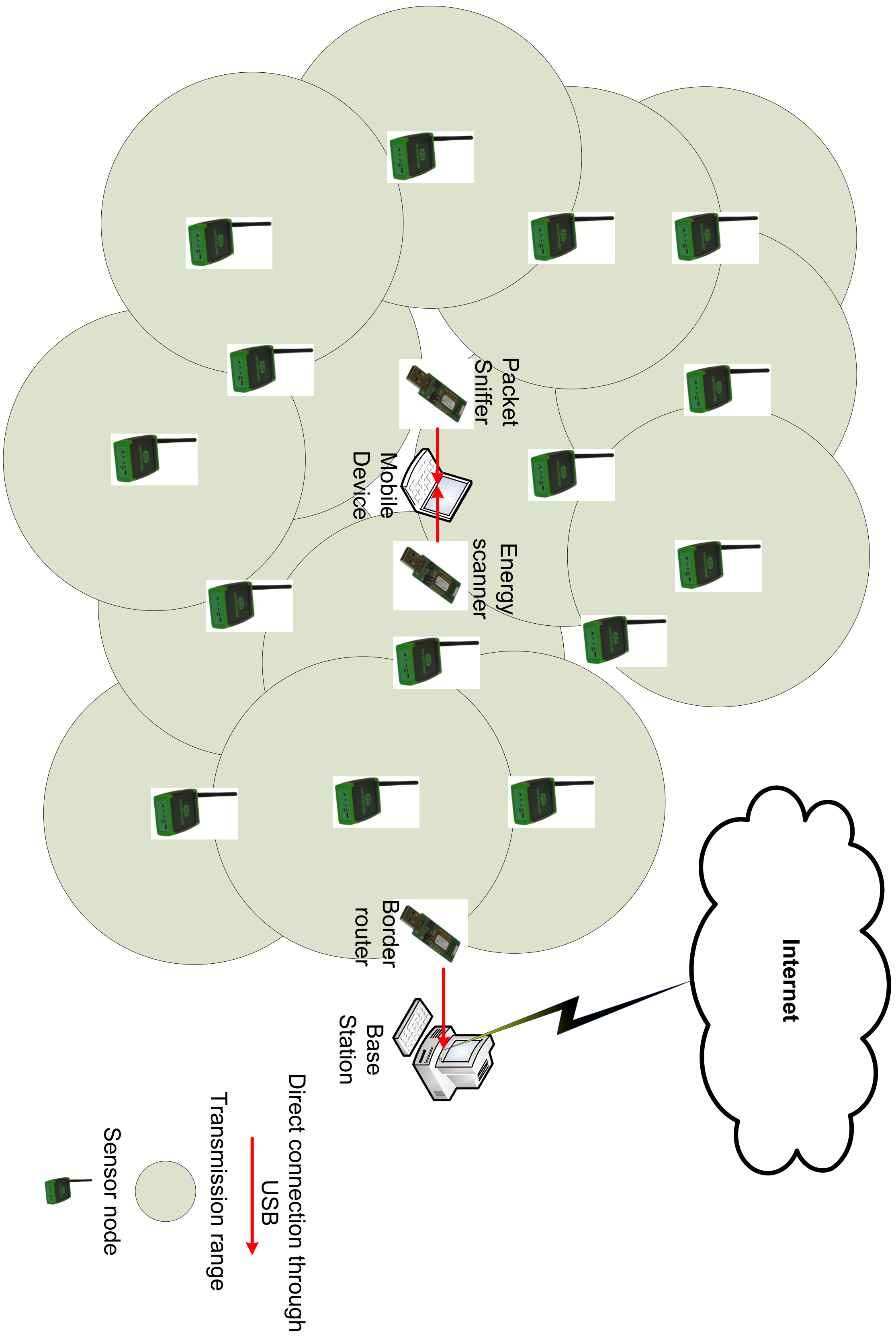

Figure 3.21: IPv6 and 6LoWPAN WSN

much more realistic behaviour than networks configured with static routing. Making a decision about the routing protocol in the network is also very important since routing protocols are responsible for the establishment of the routing tree and thus communication between the nodes. In Contiki OS the routing is handled by RPL *(there are no other routing protocols for 6LoWPAN network configurations)*. Even though RPL is the only routing protocol in Contiki *(for 6LoWPAN traffic)*, it can be configured with different OF. When configured with different OF the nodes behaviour and even the routing tree will vary. For example, when the routing tree will vary between an OF which assumes distance as the only metric and one that considers the link quality as well. Since one of the main attributes of sensor networks is lossy and unstable links, the OF used in our experiments was considering both distance and link quality as metrics. An OF such as the one mentioned can contribute in a more realistic network behaviour *(links and paths can change frequently during the experiments)*.

A WSN, designed on the above ten steps is presented in Figure 3.21.

3.11 WSN Application layer and Data Collection Server

In section 3.10, the importance of collecting the required data from the network was discussed. Moreover, the deployed network should be capable of collecting data from different scenarios with varied parameters such as data rates and number of transmitted packets per flow without the need of reconfiguring the whole network. As a result, the application layer in the nodes should be capable of dynamic reconfiguration of parameters such as transmission rates and the number of packets to be transmitted. Additionally, the data collection server must be able to collect and save in an easy to process format the desired parameters. The rest of this section describes in detail both the application layer and the data collection server used in this study.

3.11.1 WSN Application layer

The application layer used during the test-bed experiments was designed based on the characteristics of 6LoWPAN networks. Every node in the network *(except the RPL border router)*, was configured with the exact same application layer software and thus every node could become a traffic source upon request. Firstly, we divided the measured parameters in two categories:

1. parameters that need to be gathered from every node in the network. This includes the time each node spend at each of the various radio states *(radio RX, radio TX, radio idle and radio OFF)*. This time was later converted in the total energy consumption at the nodes as described in section 3.6.

2. parameters specific to the source-destination. These parameters can be the total number of packets transmitted, number of packets received and round-trip delay.

Each node was configured to operate as a UDP client. Nodes would periodically measure parameters of the first category and transmit them to the data collection server *(UDP server)*. In addition to the above parameters, each message contained a sequence number *(specific to these type of messages)*. Node crashes *(usually due to stack overflow or due to a function that keeps control for long period of time. Appendix D contains detailed information on stack overflow and MCUs based on the 8051 architecture)* could be detected through the sequence number *(will reset to 0 when a node crashes)*. In case of a node crash, the last energy measurements were added to the new energy measurements after the crash. This approach ensures as much knowledge of the status in unstable environments such as sensor networks as possible. The period between each packet transmission of this category, must be long enough not to interfere with the rest of the experiments but not too long in order to minimise the information loss in case of a node crash. During our experiments the period was configured to one minute. In our test-bed experiments, a node that only transmits this type of messages is considered idle.

Additionally to the UDP client at each sensor node, a UDP server was operating as well. The UDP server at nodes was used for the dynamic configuration of the nodes based on the experiment needs such as distance in hops, duration of the transmission in number of packets and transmission rate. To explain this in detail, each sensor node was idle until the reception of a UDP *START* packet carrying the experiment specific parameters. Upon the reception of the packet, nodes were extracting the required parameters and start transmitting packets *(CBR)* towards the data collection server. Packet transmissions could be terminated either upon the reception of a *STOP* packet or when the number of packets to be transmitted, specified in the *START* packet, reached. This way, the administrator can create various sources in different distances, transmission rates and transmission durations. *START* and *STOP* messages could be transmitted ether from the base station or from any computer connected to the 6LoWPAN network *(This can also be through the Internet)*.

The tool used for the transmission of *START* and *STOP* messages was Netcat a computer networking service for reading from and writing network connections using TCP or UDP [139, 140]. How Hex values and thus our *START* message configuration parameters can be transmitted through Netcat is shown:

```
echo -ne ‘‘\<first hex value>\<second hex value>" |
 nc   <IPv6 address> <Port number>
```

where echo is a command that places a string on the computer terminal, -n is an argument and means no new line at the end, -e is an argument and means interpret escapes and finally nc is the netcat command.

Figure 3.22 represents the state transition diagram of the sensor nodes application layer.

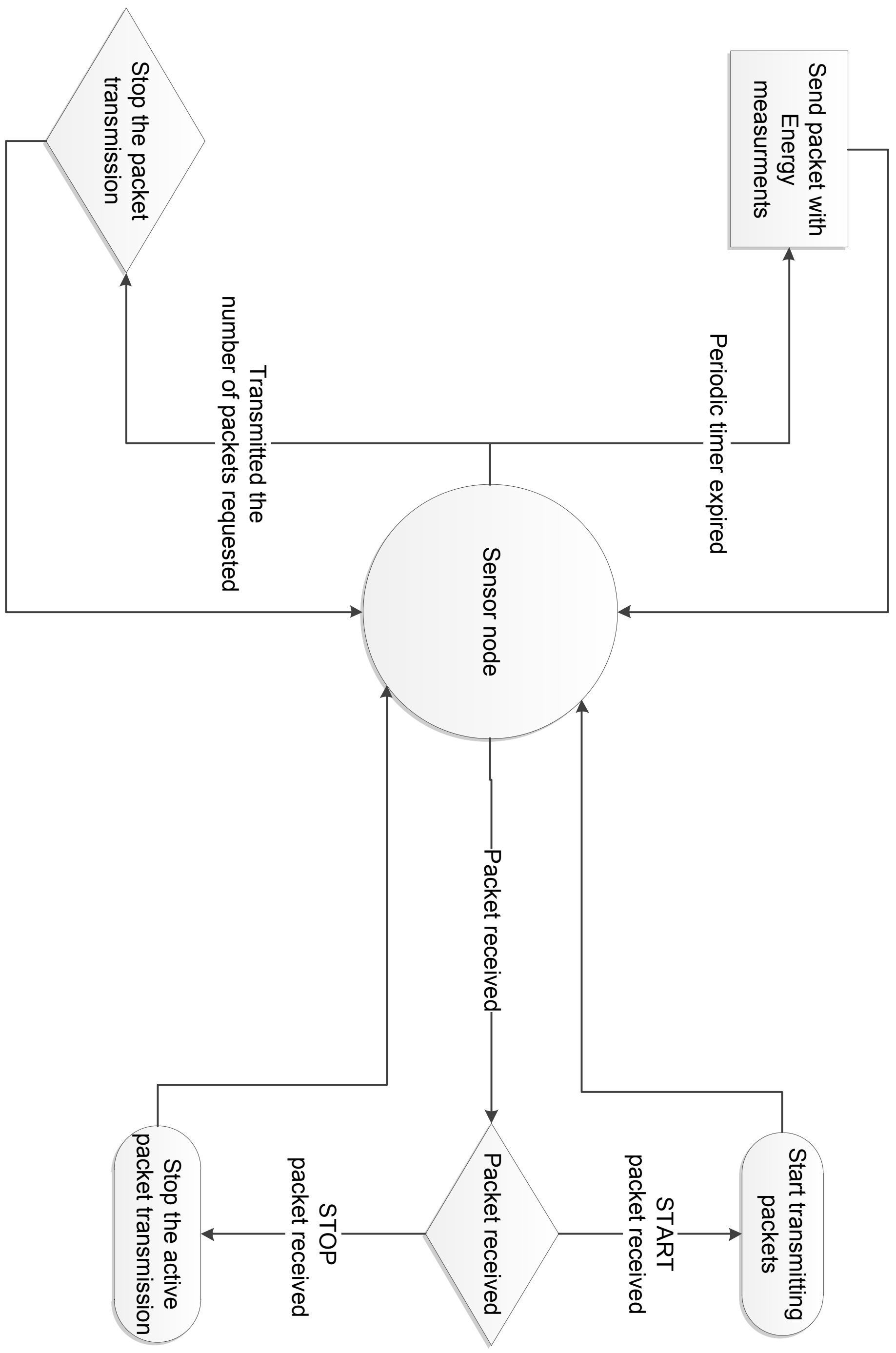
Stop the packet transmission
Send packet with Energy measurments
Transmitted the number of packets requested
Periodic timer expired
Sensor node
Packet received
START packet received
STOP packet received
Packet received
Start transmitting packets
Stop the active packet transmission

Figure 3.22: Sensor node application flow chart

3.11.2 Data Collection Server

For the collection of sensor data, an UDP server is implemented. In our experiments, the UDP server was operating in the local network directly connected to the 6LoWPAN WSN through the RPL border router. If it is required by the design of the WSN *(such as WSN deployed in remote areas)*, The UDP server can operate in any network connected through the Internet with the 6LoWPAN WSN.

As mentioned earlier, there are two types of data that the UDP server collects from the WSN: the periodic data *(energy consumption)* and the flow specific data *(packets received, packets transmitted, loss, round-trip delay and IP address of the source)*. It is very important for these two categories of data to be saved separately in an easy to process format for the ease of data processing later. Even though data from the two categories are saved separately, a mechanism that will make the linking of these two categories is required.

The implemented UDP server, makes the distinction between the different packets and the parameters carried by them and then saves the received data in two files based on the type of parameters carried within the received packets. In addition, a common time-stamp is used. The common time-stamp is later used during the processing of the data in order to link different type of data, saved in different files. The format of the saved data as well as how this data were processed is demonstrated in Appendix A.

Chapter 4

Congestion Detection (CD) in Duty Cycle and 6LoWPANs

The aims of the chapter is to identify what is the behaviour of common CD mechanisms in Contiki and IPv6 6LoWPAN networks.

4.1 Introduction

In a WSN, congestion can occur due to simultaneous packet transmission by multiple nodes as a result of an event detection (for example sudden temperature rise). Current state-of-the-art congestion control schemes operate in two phases: i) congestion detection (CD) and ii) congestion avoidance (CA). The former employs algorithms aiming to predict if the network is likely to get congested, based on observed network traffic. The latter incorporate various methods such as the ones described in section 2.3. Due to the constrained nature of sensor networks, adopting existing congestion control (CC) methods from wired networks is not suitable. For instance, it has been shown that TCP has suboptimal performance as well as high energy consumption when applied in sensor networks. This has led to the development of new CC protocols, often optimised for WSN-specific network topologies and operation (for example tree data collection). However, such optimisations can potentially harm the generality of the findings.

It has been shown that idle radio listening is a major source of energy consumption. In order to achieve energy savings and prolong the life cycle of a sensor deployment, research community members have made significant efforts on the development of radio cycling algorithms (section 3.6). Crucially, while it can have significant impact on the performance of CC, this trait of a typical WSN is often neglected when comparing various CC algorithms.

Additionally, with the entrance of IPv6 and 6LowPAN in WSN, IP architectures became one of the the most popular choice for WSN. However, the distinct features of sensor networks *(tree topology, small packets in size and simultaneous transmissions in event detection)* has created the need of alternative congestion control mechanisms for IP networks. Whether alternative congestion control mechanisms or mechanism designed for non layered architectures, can be applied to layered architectures using IPv6 and 6LoWPAN is going to be investigated in this chapter as well as how radio duty cycle affects existing congestion detection mechanisms.

There are several congestion control protocols in literature, with varied CD and CA mechanisms. These protocols and thus mechanisms are often evaluated for a limited subset of possible scenarios and parameters, although the actual space is very large. This makes it difficult to compare and evaluate the congestion mechanisms implemented in each protocol.

In literature there are many surveys for congestion control protocols in WSN [141, 142]. Additionally, a survey on congestion schemes and their CD/CA mechanisms was presented in section 2.4. In this chapter a comparative study of existing CD mechanisms is going to be presented. Such a study would provide useful information on congestion control mechanism and their design choices especially when these mechanisms have to be implemented in a layered (*IP*) WSN, aid to the development of more efficient and robust congestion mechanisms and allow widespread adaptation of congestion mechanisms by showing which design choice is appropriate for a given network scenario.

With the above observations taken into account, this chapter contribution is two-fold: i) we present results highlighting the behavior of standard congestion detection mechanisms in a multi-hop 6LoWPAN network ii) we demonstrate how RDC mechanisms affect different congestion detection mechanisms.

The rest of the chapter is organised as follows. Section 4.2 discusses the details of CD and CA mechanisms used by existing schemes as well as the CD mechanisms evaluated in this chapter. The simulation configurations are presented in section 4.3 while simulation results are presented in section 4.4. Our conclusions appear in section 4.5.

4.2 Congestion Detection Approaches

As discussed in subsection 2.4.5, congestion control schemes use different mechanisms for CD and different for CA. As a first part of the study we focus on the most used, by the WSN community, category of congestion control schemes; the rate adaptation schemes. The majority of rate adaptation schemes are using methods such as AIMD or variations thereof in order to achieve CA. This is why the focus of this chapter was on evaluating the existing CD mechanisms. Rate adaptation schemes, are mostly using mechanisms such as buffer occupancy and calculating the congestion degree by dividing the packet service time with packet inter arrival time ($CD = P_s/P_a$) in order to detect congestion. The later category can also be referred as intelligent CD as described in [63].

More specifically, buffer occupancy can be measured either with a buffer threshold *(when packets in the buffer exceed the threshold, congestion is detected)* or with a periodic buffer *(the node calculates if the buffer would overflow, if it were to receive the same number of packets in the next period P)*.

For this study, we implemented the two different ways of detecting congestion based on the buffer occupancy *(described above)* as well as the mechanism calculating the congestion degree based on $CD = P_s/P_a$. In order to evaluate the different ways to detect congestion, we tested them under the same AIMD mechanism. The implemented AIMD mechanism work as follows: when a node receives a congestion notification message, it reduces its rate to 50%. Furthermore each node increases its rate by reducing the inter packet transmission time. Let t_i be the inter packet transmission time, then the the transmission rates will be increased by reducing the t_i by t_i/δ every t_i seconds. Hence we have:

$$t_{i+1} = t_i - \frac{t_i}{\delta} \tag{4.1}$$

It is trivial to show that the transmission rate r_i will be a linear function with slope δ. The choice of δ dictates the intensity of additive increase and thus choosing a suitable value is of great significance. Let t_{min} be the lowest value for inter transmission time *(highest rate)* and t_{max} the highest after a rate reduction *(*$t_{max} = t_{min}/2$*)*. In order to avoid t_{max} to jump to t_{min} in a single step we require:

$$t_{max}/\delta << t_{max} \tag{4.2}$$

In order to achieve these requirements we set δ to:

$$\delta = \alpha \times \sqrt{t_{max}} \tag{4.3}$$

where α is a positive number greater than one *(*$\alpha > 1$*)*. The final equation will be:

$$t_{i+1} = t_i - \frac{t_i}{\alpha \times \sqrt{t_{max}}} \tag{4.4}$$

Some of the above CD mechanisms, and variations of the AIMD scheme used in this evaluation; has been used as parts of the later work. Therefore, related to the above mechanisms pseudocodes, can be found in chapter 5.

4.3 Simulation Configurations

The comparison of the above mechanisms has been conducted with COOJA *(section 3.3)*, a cross layer simulator part of the Contiki embedded operating system. The radio medium used for the experiments is called unit disk graph medium with distance loss UDGM *(allows modification of radio transmission ranges by the user)*. In order to perform our evaluation, we implemented the mechanisms as extensions of Contiki's CSMA MAC layer. The buffer size used in MAC layer is 10 packets *(total size 10 × 128 bytes)*. Furthermore, there is no buffer for incoming packets *(only outgoing packets are stored in the buffer)*. In the RDC layer, both sicslowMAC and ContikiMAC configured with a channel check rate of 8, have been used for the experiments. SicslowMAC is a non duty cycling simple protocol which creates the 802.15.4 frames and forwards them to the next node. On the other hand ContikiMAC is the default duty cycling RDC of Contiki OS *(section 3.6)*. Routing in our experiments was handled by RPL described in section 3.9. The sources were producing CBR traffic of 4 packets/s, and each packet had a 32-byte application data. The motes emulated in COOJA for our experiments are sky motes and the total duration of each simulation was 50 seconds.

The experiments were repeated under various random topologies. In order to create scenarios with high congestion, the majority of the sources were sharing the same traffic path in most of the simulation experiments. It is worthy to mention, that the above set up can lead to congestion collapse when the network is configured with ContikiMAC and a channel check rate of 8 *(8 is the default configuration of ContikiMAC)*. A higher channel check configuration could have been used in order to avoid congestion collapse but the aims of these experiments is to test the mechanisms under the default configurations in both cases *(duty cycle and not)*.

In our experiments, we had a total of 20 nodes. Among them we had 6 sources and 1 sink, while the remaining nodes were intermediates. The sources were always deployed as neighbours in order to simulate the detection of an event. For the evaluation we used two metrics: i) total packets received by the source and ii) packet loss. Since our implementation was at the MAC layer, only MAC layer transmission rates could be adjusted. Even when the sources where informed about the congestion, the application layer source rates were not adjusted. On the contrary, packets unable to be transmitted *(more than the MAC layer adjusted rate)* were discarded before transmission at the sources. Therefore, only the packets dropped after entered the network *(at least transmitted once)* was measured in the figures describing the simulation results. Even though, Table 4.2 demonstrates the total packet loss out of every packet the application attempted to transmit.

4.4 Simulation Results

Figure 4.1 displays the total received packets/second while Figure 4.2 *(on page 107)* shows the loss for each of the selected CD mechanisms *(see section 4.2)*, operating with the Sicslowmac RDC. We can observe that mechanisms using buffer occupancy performed much better than intelligent CD. We also observe that intelligent CD was less successful in receiving packets than in scenarios without congestion control. Studying Figure 4.1 *(on page 106)* and Table 4.1, mechanisms that detect congestion based on buffer state had significantly higher number of successfully received packets than both cases of no congestion control and intelligent CD. Both buffer threshold and periodic buffer performed similarly in terms of total packets received by the sink, with periodic buffer demonstrating slightly better performance. On the contrary, buffer monitoring based on threshold had lower packet loss in the network than periodic buffer. Furthermore it is noteworthy that with different period times, periodic buffer had significant differences in packet loss *(the smallest the period the lower the loss)*. More detailed results about the different performance of periodic buffer when different time periods used, can be found in [60]. Without congestion control, network performance was worse than the majority of CD mechanisms in both total packets received and packet loss. It is also notable that intelligent CD demonstrated a very poor performance during the experiments. Intelligent CD's throughput was even lower than no congestion control.

Figure 4.3 displays the total packets received while Figure 4.4 *(on page* 109 the loss for each of the selected CD mechanisms, operating with RDC *(contikimac)*. When the network was configured with a duty cycle algorithm, all of the congestion control algorithms had better performance than no congestion control. We can also observe that, similarly to the previous experiment, mechanisms detecting congestion based on the buffer state had significantly better results than intelligent CD and no congestion control. In contrast to simulations without RDC, buffer threshold based CD had higher number of received packets than periodic buffers. In addition buffer based CD had the lowest loss. With RDC big differences are noticeable at the performance of intelligent CD. In this experiment, intelligent CD performed better than no congestion control in terms of total successfully received packets. On the contrary intelligent CD had the highest loss in comparison to the rest of the CD mechanisms. Similarly to the previous experiment, without congestion control the network's performance was worse than any of the other configurations.

As mentioned previously, the aim of the compared mechanisms was to mitigate congestion towards the sources. Therefore, the number of packets lost in the network *(packets that entered the network; at least transmitted once)* is a better metric than the total number of packets lost at the application *(application attempted to transmit a packet but the packet was dropped before transmission)* for the evaluation of the above algorithms. Even though, in order to give a complete representation of our experimental results, Table 4.2 demonstrates the total packet loss at the applications.

Table 4.1: Number of successfully received packets at the sink

	Threshold	*Period 0.5*	*Period 0.25*	*Service/Arrival*	*No CC*
SicslowMAC	699	742	747	165	264
ContikiMAC	123	115	106	81	69

Table 4.2: Number of total lost packets (Application wise)

	Threshold	*Period 0.5*	*Period 0.25*	*Service/Arrival*	*No CC*
SicslowMAC	284	187	230	819	3
ContikiMAC	858	776	716	656	358

4.4.1 Performance Analysis of the Evaluated Congestion Detection Mechanisms Without Duty Cycle

When a mechanism has more successfully received packets within the same time interval, it is natural to conclude that this mechanism has higher throughput. Based on the previously discussed simulation results, we observe that intelligent CD performed poorly. In comparison to buffer state CD mechanisms, intelligent CD had over 200% lower throughput. This poor performance can be justified if we take under consideration RPL's control messages. Intelligent CD measures the congestion levels based on the received and transmitted packets and thus even control packets such as RPL control packets; that will not further contribute to buffer overflows or congestion *(only transmitted periodicaly)*, will be taken under account for the measurment of the congestion levels at the nodes. Consequently, congestion will be detected more frequently and thus the rate reduction algorithm will be triggered. It is also worthy to mention, that the more congested the network becomes, the more packets will be lost. Additionally, when the number of lost packets increases, the Link Quality Indicator *(LQI)* decreases and thus there is a higher possibility for RPL to attempt and find alternative paths. This in turn will

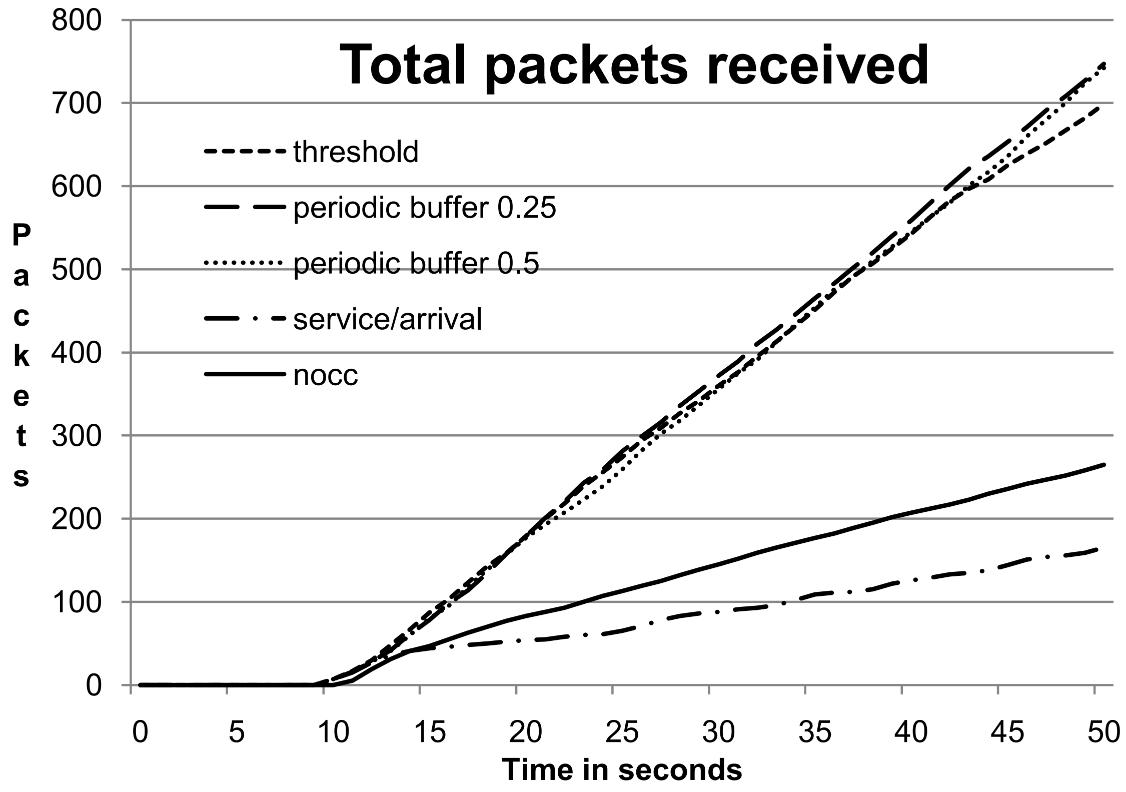

Figure 4.1: Successfully received packets without RDC (SicslowMAC)

lead to a higher overhead of RPL control messages in the network. This in turn explains why this mechanism resulted in the lowest number of received packets. On the other hand, detecting congestion frequently resulted in lower transmission rates and thus less traffic was inserted in the network by the sources. This can explain why this mechanism had no loss in the network. Based on the experiments, it is also observed that without RDC, CD based on periodic buffer had over all the most received packets and thus the highest throughput. On the other hand the highest transmission rates in the intermediate nodes resulted in higher packet loss than CD based on buffer threshold and intelligent CD. The reason behind the higher throughput of this mechanism is the higher buffer utilisation *(CD based on buffer threshold will always trigger back off when the buffer capacity reaches the threshold therefore only a percentage of the total capacity of the buffer is used. On the other hand periodic buffers can utilise buffer capacity more efficiently and avoid rate reduction messages, even when buffer occupancy levels are very high).*

4.4.2 Performance Analysis of the Evaluated Congestion Detection Mechanisms With Duty Cycle

Further analysis of the above experiments has revealed that CD (and thus congestion control) can be directly affected by RDC algorithms. Without RDC, all CD

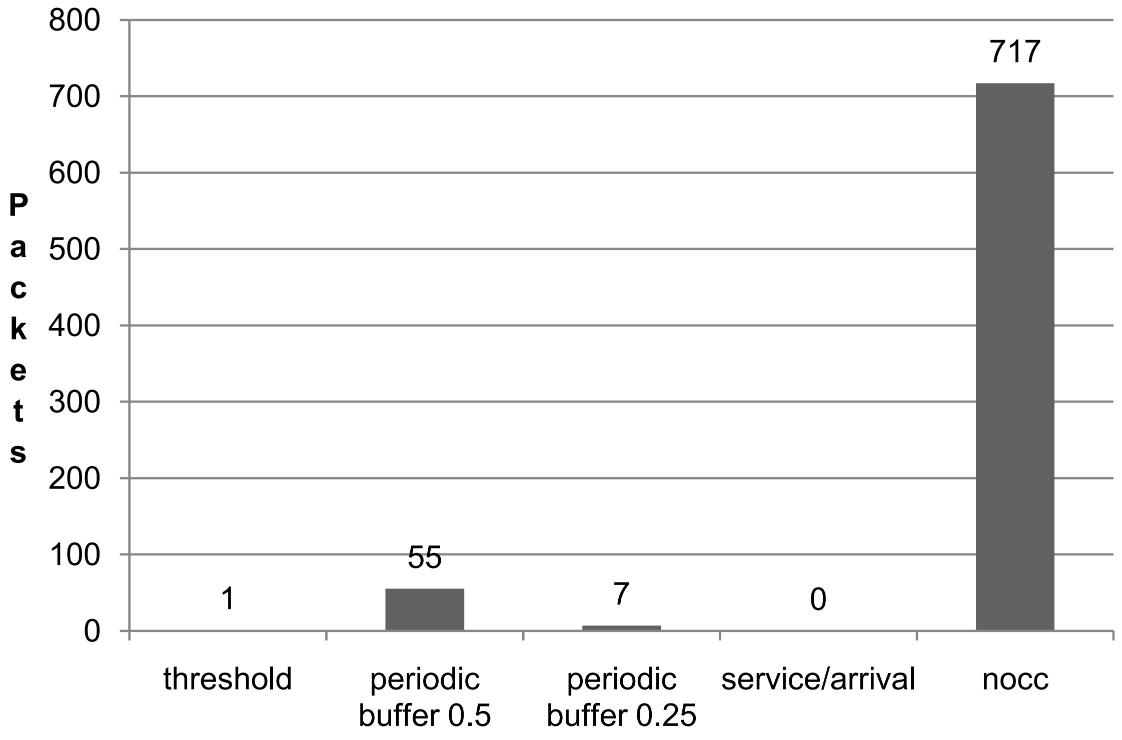

Figure 4.2: Packet Lost in the network (NO RDC - SicslowMAC - always on)

mechanisms detected and confronted congestion successfully. On the other hand, with RDC the experiment results were different, especially in the case of intelligent CD. Without RDC, Intelligent CD achieved zero packet loss but suffered from low numbers of received packets and thus throughput. On the contrary, when RDC is used the performance of this method is closer to the other CD mechanisms in terms of throughput, but at the cost of a very high packet loss rate. With RDC, nodes synchronise before transmissions or transmit trains of packets *(see section 3.6 for details)*. Additionally, nodes will frequently turn the radio off. Furthermore, since the RDC layer underlies MAC *(see details in section 3.2)*, RDC synchronisation message exchange *(or packet train packet transmission method)* happens transparently and is not taken into account by the MAC layer CD algorithm.

These distinct characteristics of ContikiMAC and thus duty cycle, can result in less frequent packet reception and thus reduce the rate at which packets are forwarded to the MAC layer. Intelligent CD is based on the periodical measurement of packet service time *(transmitted from the MAC layer packets)* over inter packet arrival time *(received by the MAC packets)*. Therefore, it is heavily affected when a radio duty cycle algorithm operates in the network since radio turn offs affect the rate of received packets. This in turn render intelligent CD incapable to successfully detect congestion. This problem could easily be fixed if ether the period in which intelligent CD measures the packet service time over packet inter arrival time was adjusted based on the duty cycle or the duty cycle is adapted

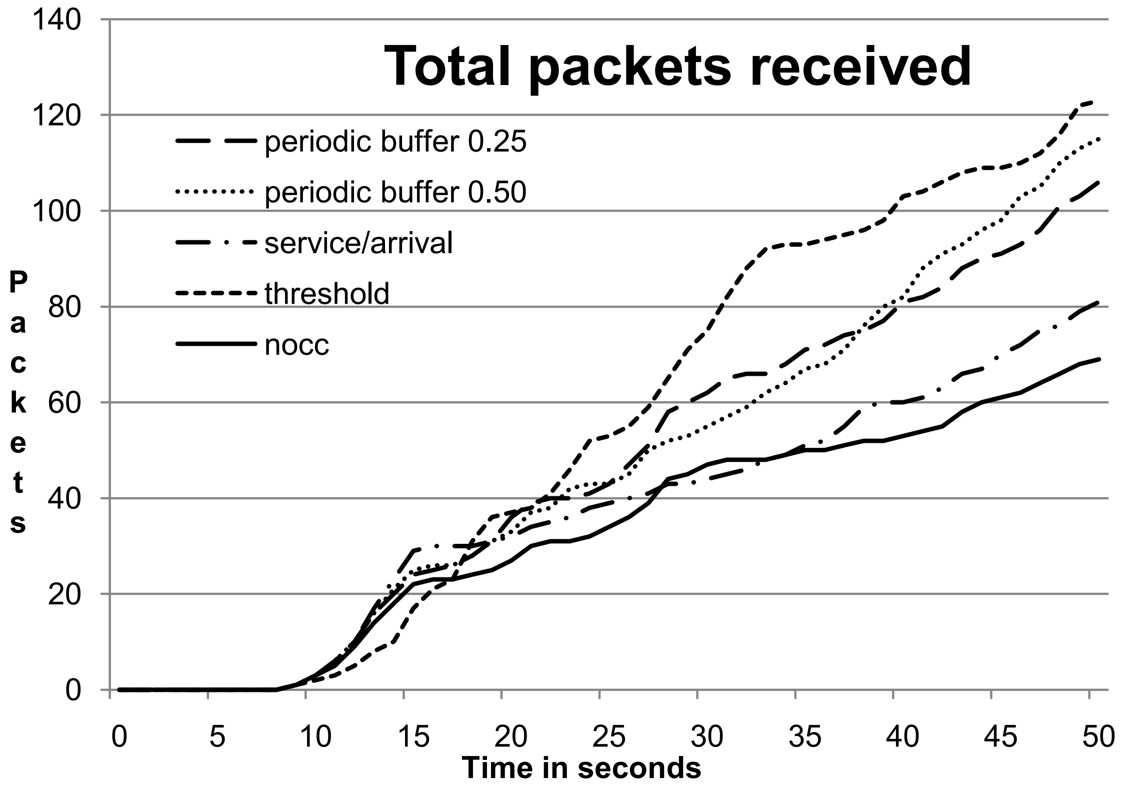

Figure 4.3: Successfully received packets with RDC (ContikiMAC)

to the period. Adjusting the duty cycle based on the congestion detection mechanism would not be the ideal option since the aim of duty cycle algorithms is to reduce energy consumption. Therefore adjusting the duty cycles based on CD would affect the energy consumption on the nodes and possibly lead to unnecessary waste of energy. On the other hand, adjusting the period of the congestion detection mechanism based on the duty cycle render the mechanism cumbersome since frequent reconfiguration would be necessary in any changes of the duty cycle in the network. This in turn can explain the poor performance of the mechanism during the simulations.

Even though CD schemes based on buffer state confronted congestion successfully both with and without duty cycle, the results were diverse. With RDC, buffer based on threshold CD resulted in approximately 20% higher amount of received packets than periodic buffers. On top of that, CD based on buffer threshold maintained insignificant packet loss. In contrast to this CD based on periodic buffer had high packet loss. The reason behind this behaviour is the same as the one described above for the poor performance of intelligent CD: the period in which periodic buffer measures it's congestion levels should be based on the channel check rate of the duty cycle to achieve the optimal results.

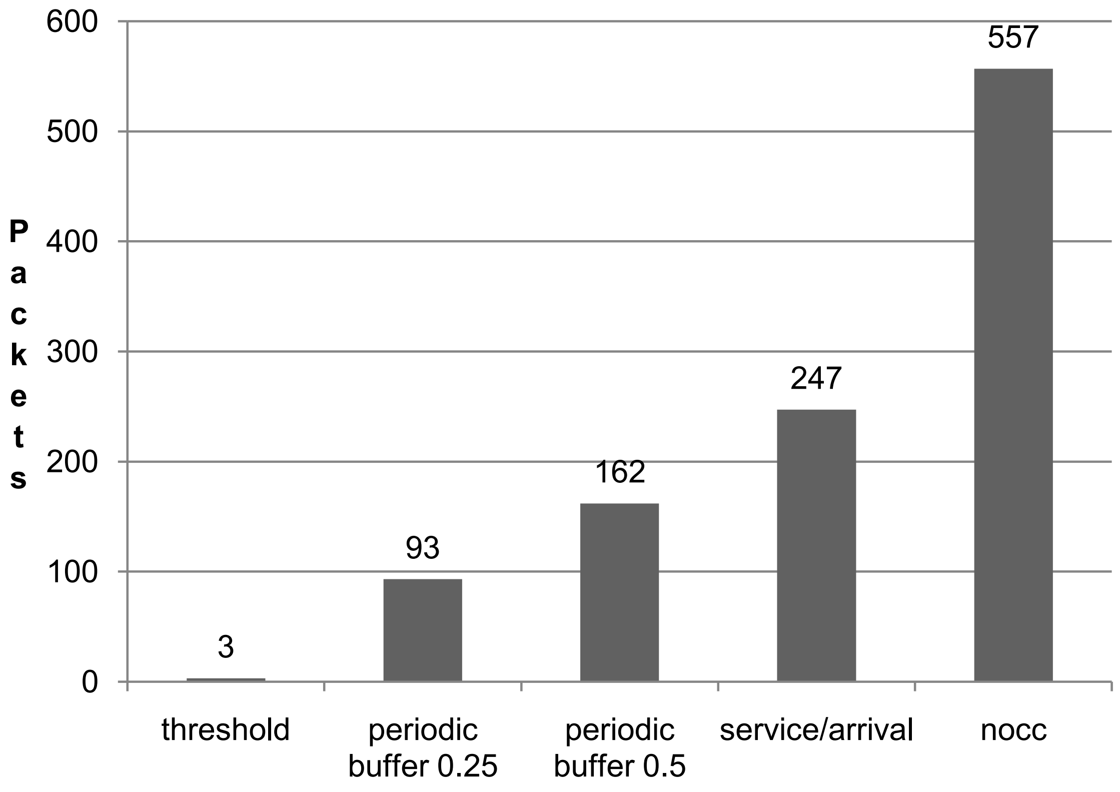

Figure 4.4: Packet Lost in the network (With RDC - ContikiMAC)

4.4.3 Overall Performance Analysis

Over all, CD based on buffer threshold is the most efficient method for congestion detection since it demonstrated the most stable behaviour in both duty cycle and non duty cycle networks; overall *(both cases of DC)* buffer based CD achieved the highest throughput and the lowest loss compared the other mechanisms. Furthermore, as observed in our experiments, intelligent CD is an inferior CD method for IPv6 sensor networks with notable constrains when the network is duty cycling*(the performance of this mechanism can be different in non IP sensor networks or in combination with different rate reduction schemes)*.

Based on Figure 4.1 and Figure 4.3, it is notable that without RDC and packet ACKs *(sicslowmac does not use any packet ACKs)*, our network demonstrated a six-fold throughput increase under heavily congested scenarios. This results was based on the default configuration of duty cycle in Contiki with channel check rate of 8. The more the channel check rate is increased the smallest the difference will become between duty cycle and non duty cycle networks. On the contrary, higher duty cycles lead to significantly higher energy consumption *(more detailed results about the energy consumption with different configurations of channel check rate can be seen in chapter 6)*. Since sensor networks are most of the time idle, it is recommended to be configured with the default *(usually the one of lowest configurations)* channel check rate configurations.

In this study, it is demonstrated, that WSN can achieve significantly higher throughput and lower loss without duty cycling. On the other hand it is not realistic to assume that a self powered wireless sensor network operates without a RDC mechanism. Efficient RDC can extend a deployment's lifetime expectancy by orders of magnitude, up to years.

Lastly, the decision of which mechanism is better totally depends in the demands, configuration of the WSN. If the sensor network requires high throughput *(no RDC)* and can afford a slightly higher packet loss, periodic buffer can be the best option, while in the remaining cases CD based on buffer threshold can successfully detect and confront congestion.

4.5 Summary and Discussions

In this chapter, we conducted a simulation-based comparative study concerning the performance of multiple CD mechanisms in 6LoWPAN sensor networks. We implemented the mechanisms for the Contiki operating system and tested them under different network conditions and protocols such as with and without duty cycle.

Our main conclusion of interest is that some of the existing congestion detection mechanisms *(packet service time / packet arrival time)* perform poorly in 6LoWPAN sensor networks especially under the presence of a duty cycle algorithm. Therefore congestion control protocol architects must carefully choose the methods and mechanisms incorporated in their design and not just import existing methods.

Moreover, it is shown that the performance of CD algorithms is directly influenced by the underlying radio duty cycling mechanisms *(this applies to congestion control mechanisms designed as parts of the lower network layers).* This chapter is a significant step towards understanding the performance of various CD mechanisms in a 6LoWPAN environment, with or without duty cycle, and can serve as a basis for future network designers, when facing the decision of which CD mechanism to adopt for their deployment.

The results of this section, had a significant impact on the design of the proposed algorithms in this thesis. More specifically, DCCC6's CD mechanism and design has significantly been inspired from this work.

Chapter 5

DCCC6: Duty Cycle-Aware Congestion Control (CC) for 6LoWPANs

The proposed mechanism is influenced by the results of chapter 4 and the most commonly used type of congestion control schemes for WSN the rate adaptation schemes.

5.1 Introduction

In Wireless Sensor Networks, congestion can cause a number of problems including packet loss, lower throughput and poor energy efficiency. These problems can potentially result in reduced deployment lifetime and under-performing applications. This has led to several proposals for congestion control schemes for sensor networks. Furthermore, the WSN research community has made significant efforts towards power saving MAC protocols with Radio Duty Cycling (RDC). However, chapter 4 revealed that a great number of the existing congestion control mechanisms have neglected the presence of a RDC scheme in their design.

The most important challenge in this work, is to design a protocol- and topology-independent CC mechanism through which each node can locally detect congestion, and signal all relevant nodes to fairly adapt their rates. Furthermore, a mechanism should lead to similar results regardless of the choice of duty cycling mechanism (or lack thereof). In general, the traffic communication patterns of IPv6 WSNs are arbitrary. The proposed mechanism should be able to maintain its performance, detect congestion and fairly adapt node transmission rates under any communication pattern.

In this context, this chapter contributes a new CC scheme for Duty Cycle and IPv6 over Low power Wireless Personal Area Networks *6LoWPAN* sensor Networks—*DCCC6*. DCCC6 detects the presence of duty cycling and adjust its operation accordingly. Both simulations and a testbed are used for the evaluation of DCCC6. The experimental results have shown that DCCC6 achieved higher goodput and lower packet loss than previous works. Moreover, simulations show that DCCC6 maintained low energy consumption, average delay times and achieved a high degree of hop-by-hop fairness.

The remainder of this chapter is organised as follows: In section 5.2 the design considerations of DCCC6 are discussed. In section 5.4 the detailed design of DCCC6 is presented. Section 5.4 presents a detailed comparison of DCCC6 with other congestion control schemes evaluated with simulations. Section 5.5 discusses the test-bed experiments while a comparison between the simulation and the test-bed results is presented in section 5.6. Finally section 5.7 has DCCC6's performance evaluation summary and general discussions about the scheme.

5.2 DCCC6 Design Considerations

Contrary to traditional, application-centric WSN design, a 6LoWPAN may be the host of a variety of applications (e.g. network management, data collection, Constrained Application Protocol services) simultaneously. If this is the case, then:

1. Packets from different applications will have different sizes, priorities and possibly different destinations
2. Traffic will be bidirectional since most of these applications require constant exchange of packets with the destination
3. Applications may use different protocols for communication *(udp or tcp)*
4. Large packets may get fragmented by the 6LoWPAN adaptation layer.

Considering the above characteristics the CC protocol should:

a) be protocol-independent

b) be topology-independent

c) forward packets based on their priority.

d) implement a traffic-aware congestion detection (CD) method.

As mentioned in section 3.6, duty cycling nodes are switching their radio state to on and off for energy saving. A node will also turn its radio off when a collision is detected, as well as after a packet acknowledgment is received. As network density increases, so does collision probability. When a collision occurs, a back off retransmission time for the packet is calculated randomly *(traditional CSMA)*. Even with the most advanced synchronisation algorithms, the existence of collisions makes perfect synchronisation between nodes very challenging. In order to address it, existing protocols transmit link layer frames repeatedly until they receive a successful acknowledgment *(train of packets like ContikiMAC does)*, or until a collision is detected. This works efficiently for link layer unicasts; however broadcast packets never get acknowledged and are thus inefficient [143].

Chapter 4 have shown that RDC highly affects the performance of CC schemes. The majority of CC protocols operate by control data exchange among neighbouring nodes, either with broadcast frames or by piggy-backing information inside acknowledgment frames. However, under the presence of RDC, link layer broadcasts are inefficient. Furthermore, with modern 802.15.4 radio transceivers, acknowledgments are generated automatically by the hardware and it is thus impossible to modify their content. The only way to tackle this problem is by disabling this feature and by re-implementing it by software, thus incurring costs in terms of processing time, code size and complexity. For those reasons, there is need for further investigation into CC algorithms for IPv6 WSNs.

5.3 Implementation of DCCC6

DCCC6 is made up from two components: i) a congestion detection mechanism and ii) a rate adaptation scheme.

The state transition diagram of DCCC6 can be seen in Figure 5.1 while a detailed analysis of DCCC6 and its mechanisms follows.

5.3.1 Congestion Detection

Previous works on Congestion Detection (CD) have demonstrated that measuring queue occupancy is one of the most efficient and accurate methods [47], [chapter 4].

DCCC6 maintains a single packet queue for all flows passing through the node. In order to detect congestion, it adopts a dynamic threshold similar to the one used in [53]. Queue occupancy is monitored on a per incoming packet basis. Congestion is triggered when the number of messages in the buffer exceeds a threshold. By inspecting incoming traffic, the congested node identifies the source of the problem and sends a congestion notification. In order to avoid notification storms, the congested node then dynamically adjusts the threshold. This process goes under multiple iterations until the queue overflows or starts to drain. Figure 5.2 demonstrates how DCCC6 dynamic buffer operates.

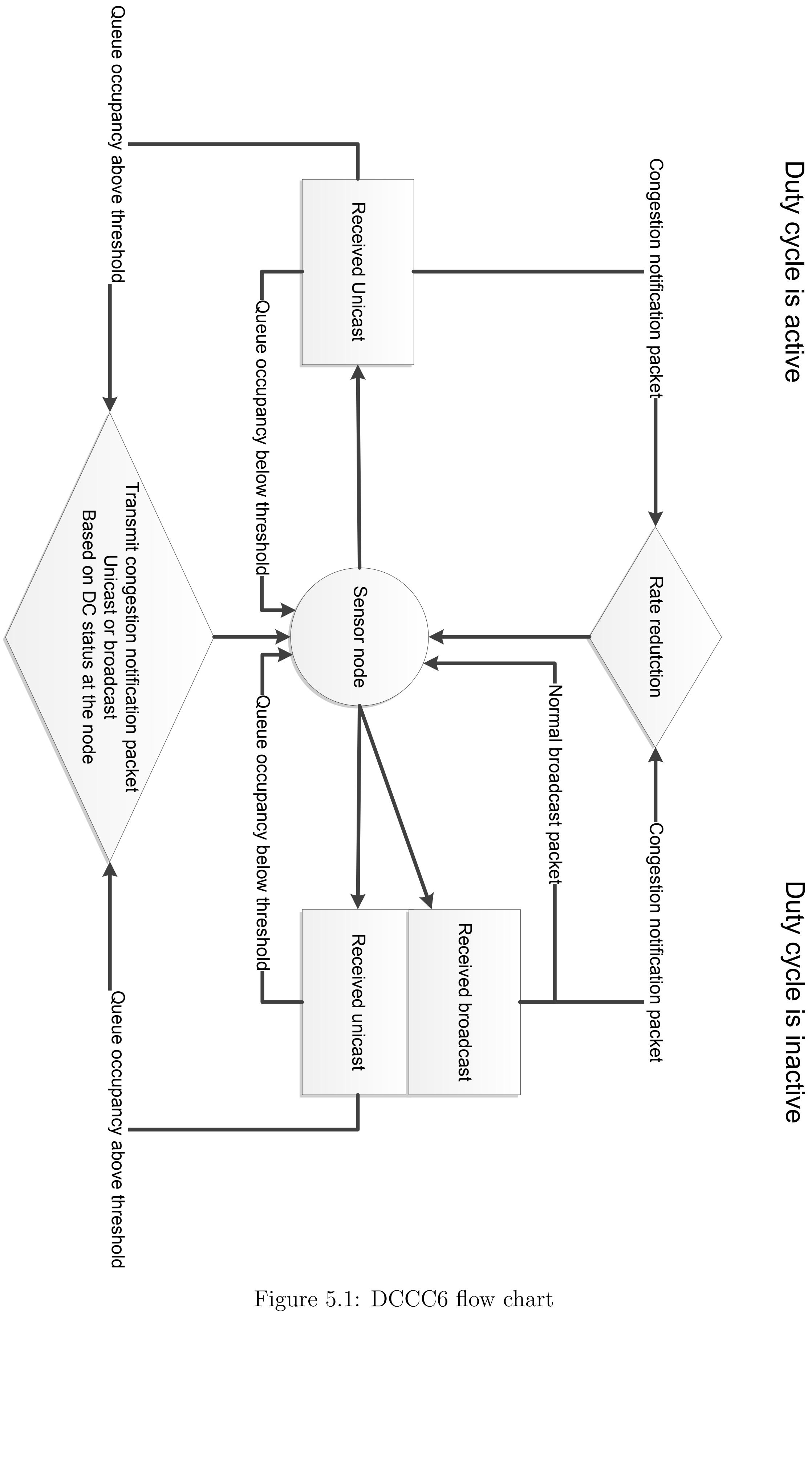
Duty cycle is active
Duty cycle is inactive
Congestion notification packet
Rate redutction
Congestion notification packet
Received Unicast
Normal broadcast packet
Received broadcast
Sensor node
Received unicast
Queue occupancy below threshold
Queue occupancy below threshold
Transmit congestion notification packet
Unicast or broadcast
Based on DC status at the node
Queue occupancy above threshold
Queue occupancy above threshold

Figure 5.1: DCCC6 flow chart

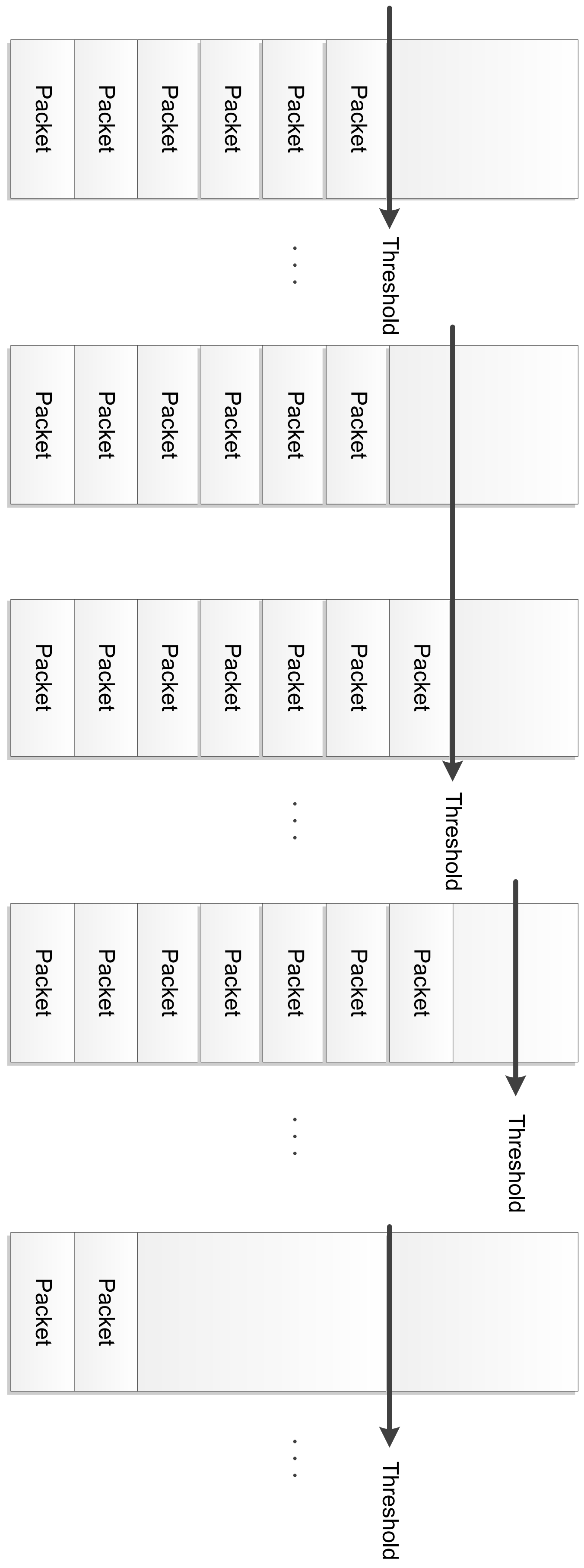

Figure 5.2: Dynamic buffer in DCCC6

5.3.2 Rate Adaptation

DCCC6 can detect the presence of duty cycling and adjust its behaviour accordingly. This can be achieved by requesting duty cycle specific information from the RDC layer. When an RDC scheme is in operation, notifications are sent inside unicast frames in order to bypass RDC broadcast inefficiency. Conversely, if radios are always on, DCCC6 reverts to notifications with a broadcast link layer destination.

When using unicast notifications, different recipients will receive a different number of rate reduction messages. Thus, a traditional additive increase and multiplicative decrease *(AIMD)* approach, would lead to unfairness. For this reason, DCCC6 uses a modified rate adaptation scheme, achieving fair bandwidth allocation. Figure 5.3 illustrates a qualitative comparison between traditional AIMD and the adaptation scheme adopted by DCCC6.

5.3.2.1 Rate Reduction

Let t_i be the interval between two successive frame transmissions, with a maximum value of t_{max}. Each time a congestion notification is received, transmission rate must decrease. We achieve this by increasing inter-packet interval t_i by α. For lower transmission rates, reductions are smaller: α itself is decreased as t_i increases. Thus:

$$\begin{aligned} t_{i+1} &= t_i + \alpha \\ &= t_i + \frac{\gamma \times \sqrt{t_{max}}}{\sqrt{t_i}} \end{aligned} \tag{5.1}$$

where γ is a factor used to control the slope with $\gamma > 1$.

5.3.2.2 Rate Increase

Transmission rates increase periodically by reducing t_i by t_i/δ every t_i seconds. Hence we have $t_{i+1} = t_i - \frac{t_i}{\delta}$. Since δ will decrease when t_i increases, it is trivial to mention that the transmission rate r_i will be a non linear function. The choice of δ dictates the intensity of rate increase and thus choosing a suitable value is of great significance. Let t_{min} be the lowest value for inter transmission time *(t_{min} represents the highest rate, and it is related to the duty cycle configuration in the network. How t_{min} is related to duty cycle can be seen in Table 5.1).* In order to avoid t_{max} jumping to t_{min} in a single step we require $t_{max}/\delta << t_{max}$. In order to achieve these requirements we set δ to

$$\delta = \frac{\beta \times t_i \times \sqrt{n_i + 1}}{(\epsilon \times \sqrt{t_{min}}) - \sqrt{t_i}} \tag{5.2}$$

where β is greater than one. The higher the value of β, the lower the function slope. On the other hand ϵ prevents the denominator taking the value of 0 after a congestion notification message is firstly received. Therefore, the value of ϵ is related to the CCR and t_{max}. Table 5.1, demonstrates how ϵ and t_{min} change for the different duty cycle configurations *(for the different Channel Check Rates)*. The number of active child nodes is represented with n_i. The final equation will be:

$$t_{i+1} = t_i - \frac{\beta \times t_i \times \sqrt{n_i + 1}}{(\epsilon \times \sqrt{t_{min}}) - \sqrt{t_i}} \tag{5.3}$$

With this design child nodes may receive an unequal number of rate reduction messages. This ensures a high degree of fairness. Nodes with lower rates will increase the rate faster than nodes with higher rates and the opposite is observed in case of rate reduction.

Table 5.1: An example of the relation between duty *(expressed as CCR)* and the variables: ϵ, t_{min} for t_{max} = 256t *(clock ticks)*

CCR	ϵ	t_{min} *(time in clock ticks)*
N/A *(No DC)*	16.1	1
8	4.1	16
16	5.7	8
32	8.1	4
64	11.4	2

The values of ϵ displayed in Table 5.1,

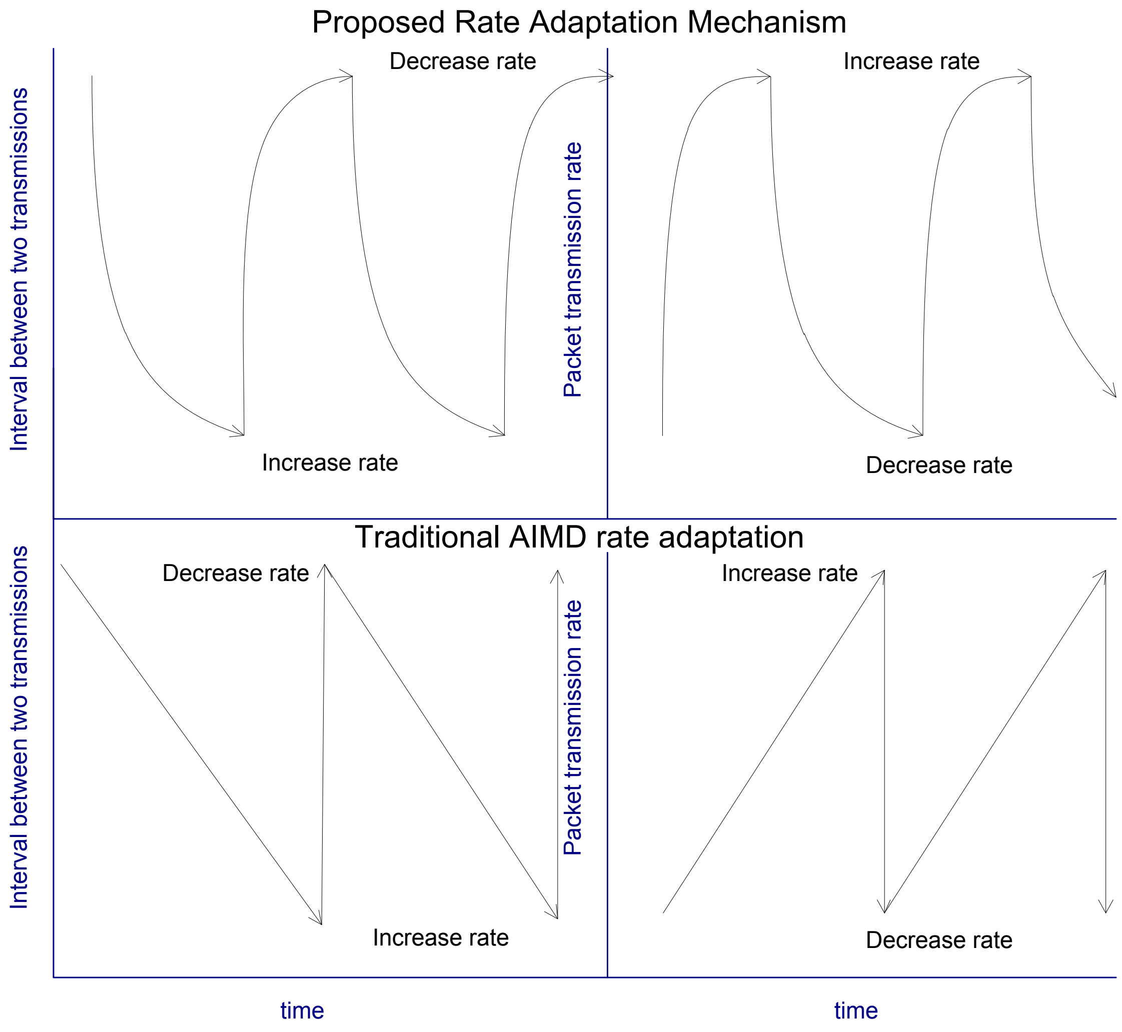

Figure 5.3: Comparison between traditional AIMD and proposed rate adaptation

5.4 Simulation Experiments

5.4.1 Simulation Configurations

Simulations aimed at evaluating behaviour in a heavily congested, duty cycled WSN. We used the cooja simulator, distributed as part of the Contiki OS. All features of DCCC6 described in section 5.3 have been implemented as a new MAC layer in the Contiki embedded operating system. The packet queue can hold a maximum of 10 outgoing packets *(10 × 128 bytes)*. As an addition, when the queue were full the oldest packet in the queue was droped and replaced by the new ones *(traditionally in Contiki when the queue is full, incoming packets are transmitted unreliable similarly to broadcasts. The traditional approach can some times give a boost to goodput but in multiple cases such as fragmentation it leads to complete network collapse or out of order packet reception)*. At the RDC layer we used contikimac with a channel check rate of 8. Contikimac [143] uses 802.15.4

radio acknowledgements (ACK) and turns the radio off for energy saving. Routing in our experiments was handled by RPL described in section 3.9. Cooja's Unit Disk Graph Medium (UDGM) was used to emulate the radio environment.

The experiments were repeated under various random topologies with similar results. In order to create scenarios with high congestion, the majority of the sources were sharing the same traffic path in most of the simulation experiments. For each experiment we used a total of 25 emulated sky motes, positioned randomly (6 sources, 1 sink, 18 intermediate nodes). The sources were producing CBR traffic of 6 packets/s, and each packet had a 32-byte payload. Each experiment was repeated 15 times with a new random seed per iteration. For each experiment we recorded the following metrics:

1. Goodput as total number of packets received by the sink,
2. Packet loss,
3. End to end delay (from source to sink) *(the delay measured was from multiple sources of the same distance from the base station)*,
4. Energy consumption,
5. Jain's fairness index [144] *(all sources were transmitting in the same rate, packets of the same size and were at the same distance from the base station in the experiments that fairness was measured).*

5.4.2 Simulation Results

Table 5.2: Packet Loss *(loss due to reception of corrupted packets is not calculated)*

	HCCP [60]	AFA [52]	IFRC [53]	DCCC6	CSMA
Packet loss at source nodes *(MAC layer)*	4784	6561	6528	5444	1142
Packet loss at intermediate nodes *(MAC layer)*	1223	0	32	92	5032
packet loss max number of retransmissions *(MAC layer)*	1398	1163	249	829	979
Total packet loss *(MAC layer)*	7405	7724	6809	6365	7153

Table 5.3: Jain's Index Fairness

	HCCP [60]	AFA [52]	IFRC [53]	DCCC6	CSMA
Average	0.799	0.681	0.742	0.759	0.728
Standard deviation	0.099	0.104	0.068	0.049	0.039

Table 5.4: Diversity Energy Consumption

	HCCP [60]	AFA [52]	IFRC [53]	DCCC6	CSMA
Standard deviation	1.185	0.545	0.318	0.343	0.483
Energy consumption per successful packet(joules)	0.00151	0.00195	0.000824	0.000813	0.001148485

In Figure 5.6, can be observed that HCCP's goodput was one of the lowest, while it maintained the highest energy consumption (Figure 5.10) and the second highest packet loss (Table 5.2). The high energy consumption of HCCP was expected since nodes with this mechanism are periodically transmitting broadcast packets in order to share their congestion states. Therefore the additional, periodical overhead cause by this packets is responsible for the mechanism's high energy consumption. From Table 5.3, it can be observed that this scheme had the best average Jain's index fairness. If we further study Figure 5.8, we can observe that during smaller intervals of time the Jain's index fairness of this protocol was very low. On the contrary, in Figure 5.7 can be observed that this scheme maintained a low and very stable delay during the experiments. This is expected since HCCP demonstrated low goodput, by mitigating congestion towards the sources and keeping the transmission rates low in the network. This in turn resulted in less occupied packet queues and thus lower delay.

The reason behind HCCP's poor performance is the periodic broadcasting of frames, used in order to exchange congestion information. As discussed in section 5.2 broadcasts are very inefficient with ContikiMAC. Additionally, if a node does not receive the broadcast, it will not update its congestion state and this will result in repeated broadcasts transmissions *(notification storm).*

With AFA, a sensor will forward a packet only if its parent has enough space in the packet queue. As seen in Table 5.2 this protocol had no packet loss at the intermediate nodes. Furthermore, due to it's nature, AFA's average energy

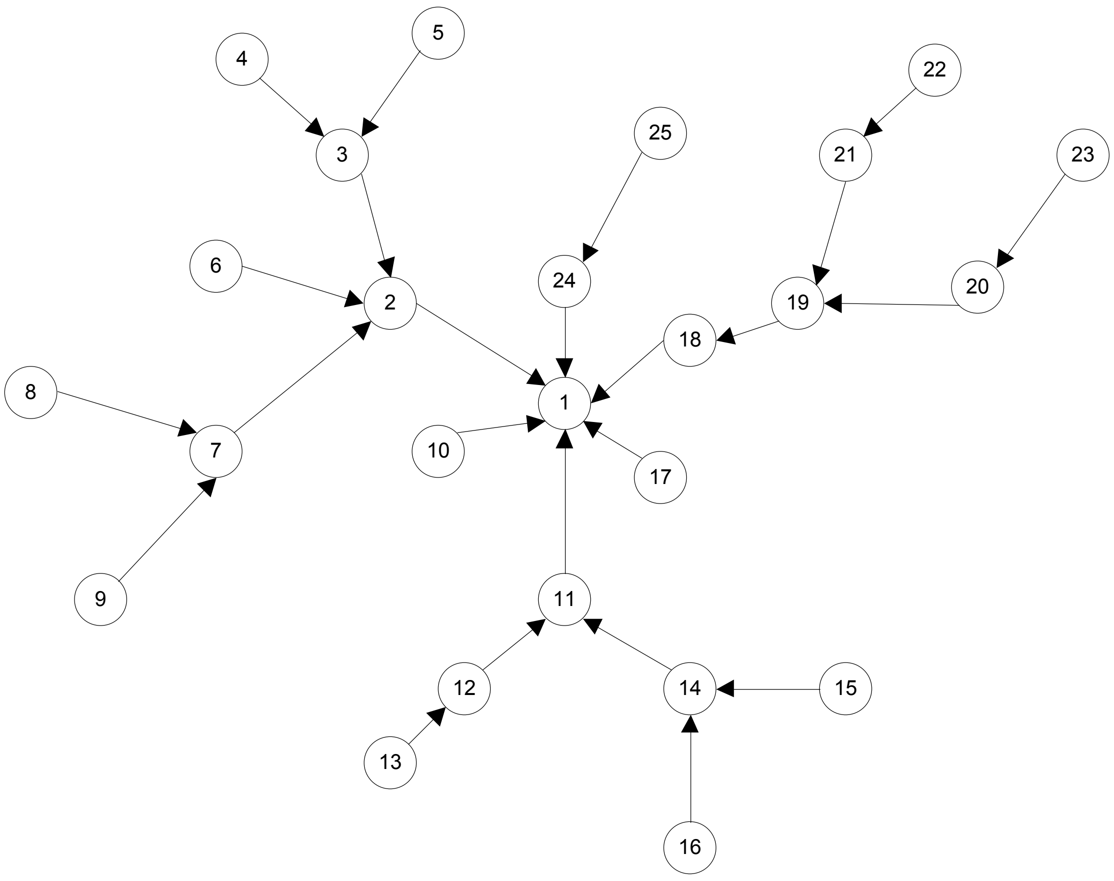

Figure 5.4: Routing tree used in our simulations experiments

consumption was the lowest out of the compared schemes. This is expected since nodes will remain idle after the reception of a congestion notification message and until the reception of a *START* transmission message. On the other hand, since many nodes may remain idle, the energy consumption was not balanced between the nodes (Figure 5.10 and Table 5.4.2). As discussed earlier, radio acknowledgments are generated by the hardware and thus it is not possible to modify them in order to carry congestion information. Therefore for the implementation of AFA broadcast messages were used for the propagation of *START* and *STOP* transmission messages. During section 5.2 was explained that broadcasts are unreliable when a duty cycle protocol is operating. Due to the previously discussed characteristics of ContikiMAC, used as the duty cycle algorithm in our experiments; informing the child nodes to start transmitting packets again was significantly delayed. This in turn lead to longer than intended idle times at some nodes. This problem could be confronted if the software acknowledgments were implemented instead of the hardware or unicast messages were used for the congestion notification *(START - STOP messages)* instead of broadcasts. On the other hand, both software acknowledgments and unicast congestion notification messages have their own draw backs. By implementing software acknowledgments, the interval between packet transmissions would noticeable increase and thus the scheme's performance. Due to the nature of unicasts *(always inform one child)*, the later solution can lead to cases where not all child nodes receive a packet signaling the start of transmission and thus unfairness in the network. The above observations can explain why AFA had the lowest goodput and the lowest degree of fairness (Figure 5.5 and Figure 5.9).

Studying Figure 5.10 and Table 5.4.2, we can conclude that IFRC had the most balanced energy consumption between nodes, while its average energy consumption was the second lowest. Furthermore, it managed to successfully confront congestion in a duty cycling network and thus maximised goodput. In addition, Table 5.2 shows that IFRC maintained low packet loss *(compared to most of the schemes)* and a very good degree of fairness. IFRC maintained high transmission rates at the intermediate nodes and thus the packet queues were built up more than some of the other schemes in the simulations. The built up packet queues is the reason for IFRC's high delay times (Figure 5.7).

If we further analyze Figure 5.5, Figure 5.6 and Table 5.2, it is observed that the proposed scheme DCCC6, had the best performance in terms of goodput and packet loss with 18% higher goodput and 7.4% lower loss than IFRC which had the second best performance. Since both DCCC6 and IFRC use a similar CD mechanism, DCCC6's CA mechansim is responsible for it's higher goodput. This can be justified if we take under consideration that traditional AIMD *(used in*

IFRC) mechanisms can lead to more than desired rate reductions in case of consequent reception of congestion notification messages. On the contrary, DCCC6's CA mechanism tries to maintain ideal rate's and rate stability between the flows. Even though, DCCC6's overall energy consumption was not the lowest *(second lowest)*; if the energy consumption is calculated per successfully received packets at the base station, DCCC6 is the most energy efficient out of the compared protocols (Figure 5.10 and Table 5.4.2). This is expected since DCCC6 had the highest goodput and it's mechanisms trying to always achieve the optimal transmission rates at the nodes. Further analysis of Figure 5.7 shows that the packet delay of DCCC6 was lower than IFRC and AFA but higher than HCCP and CSMA. DCCC6 is trying to fully utilise the packet queue *(during congestion, all nodes are trying to maintain the highest posible rate which leads to slow drain of the queues)* in the intermediate nodes and thus the delay increases. The Fairness achieved by the proposed scheme was overall the highest. This is expected since DCCC6's rate adaptation mechanism is designed to distribute similar transmission rates at the various-active child nodes. It is worthy to mention that DCCC6 managed not only to maintain high overall fairness but consistent high fairness during any instant of the simulations. Moreover, DCCC6 had the lowest diversity in the results of the different experiments (Figure 5.8, Figure 5.9 and Table 5.3).

Figure 5.4 illustrates the routing tree forged by RPL protocol during simulations.

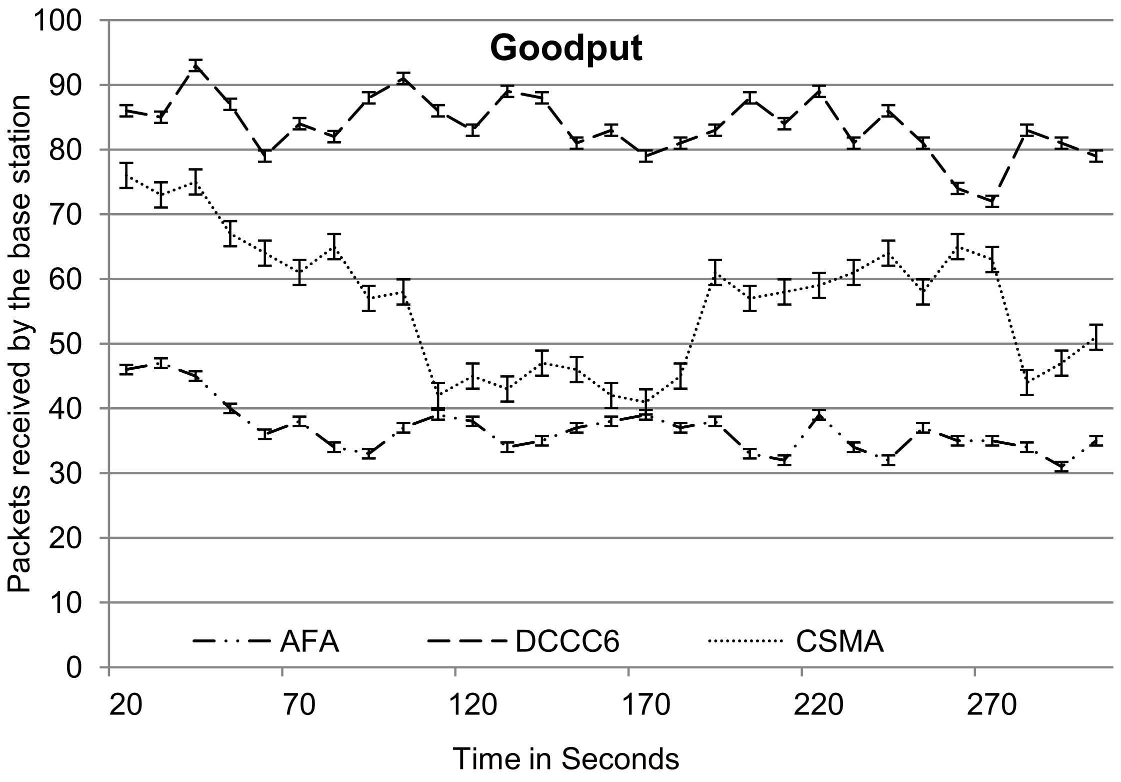

Figure 5.5: Successfully received packets at the sink

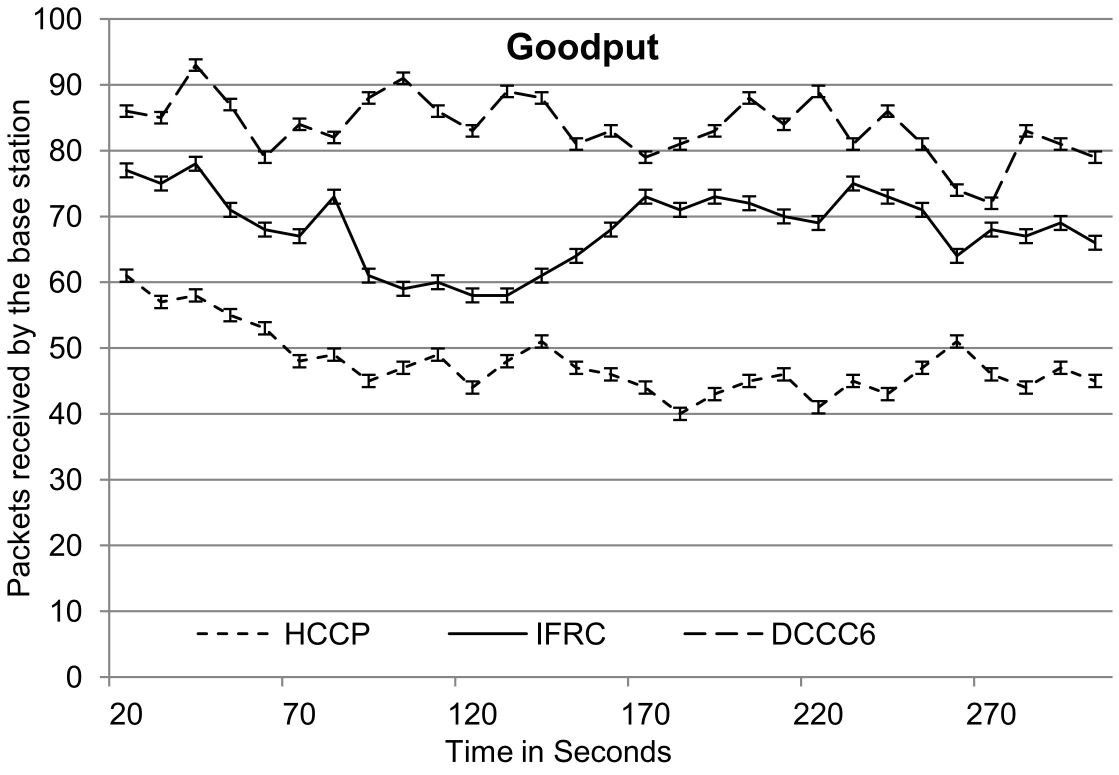

Figure 5.6: Successfully received packets at the sink

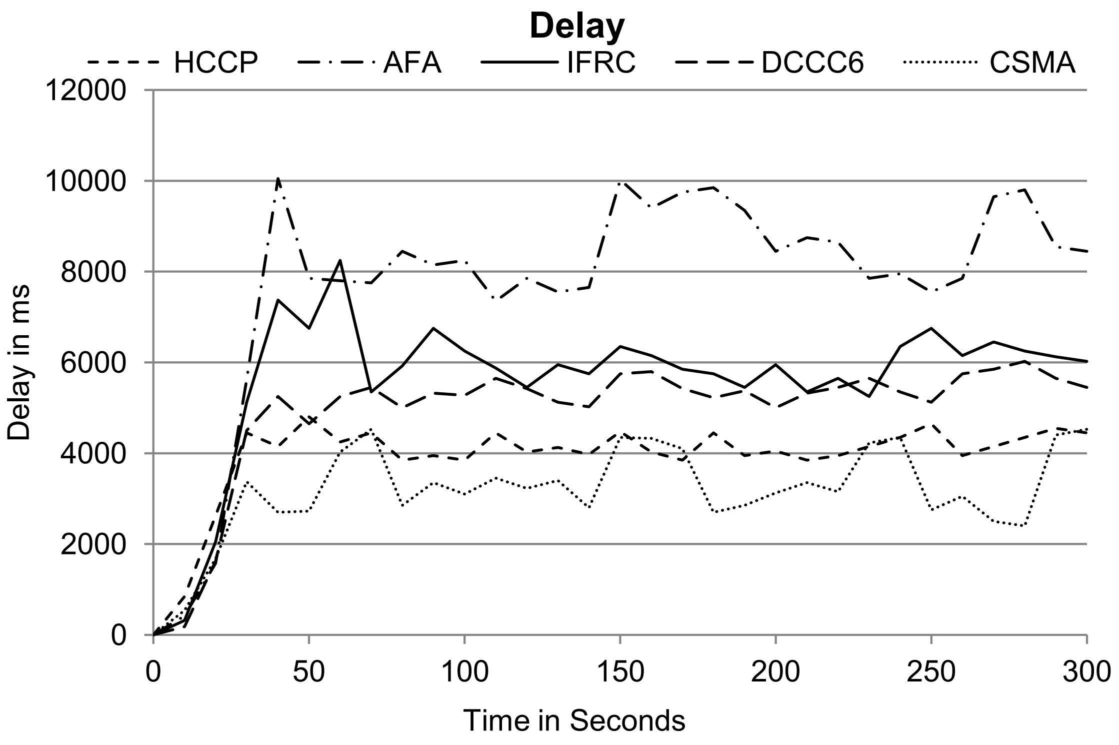

Figure 5.7: Packet delay

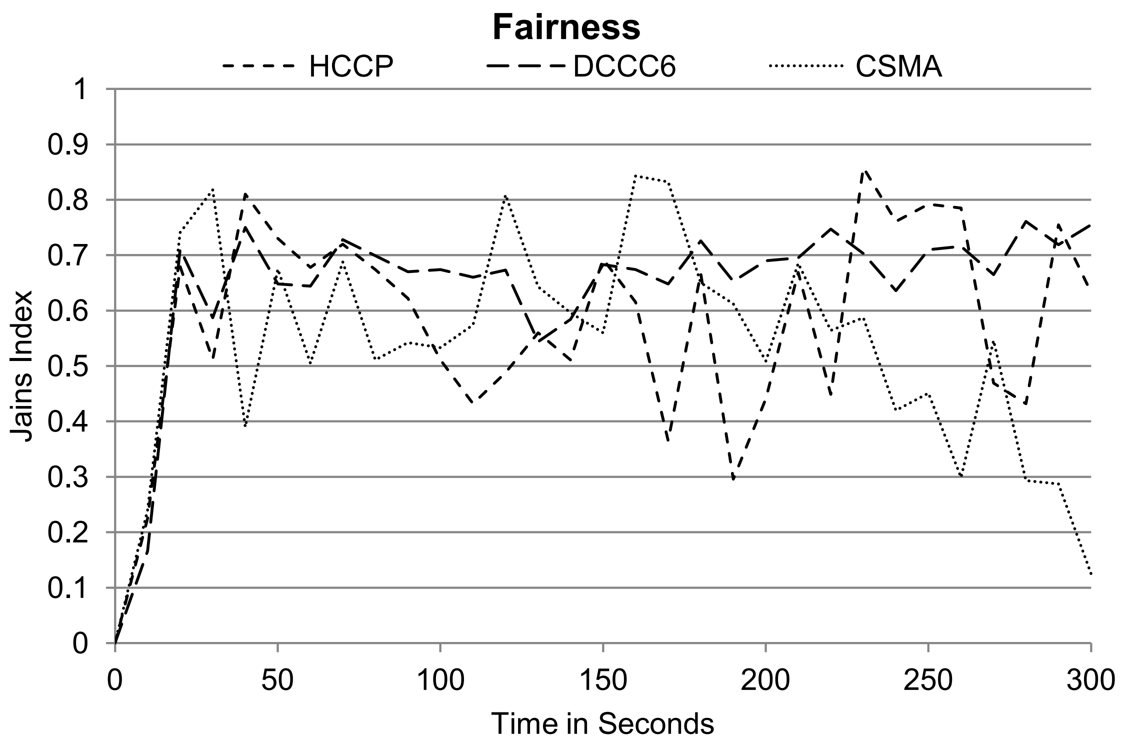

Figure 5.8: Jain's Index fairness over 10 second intervals

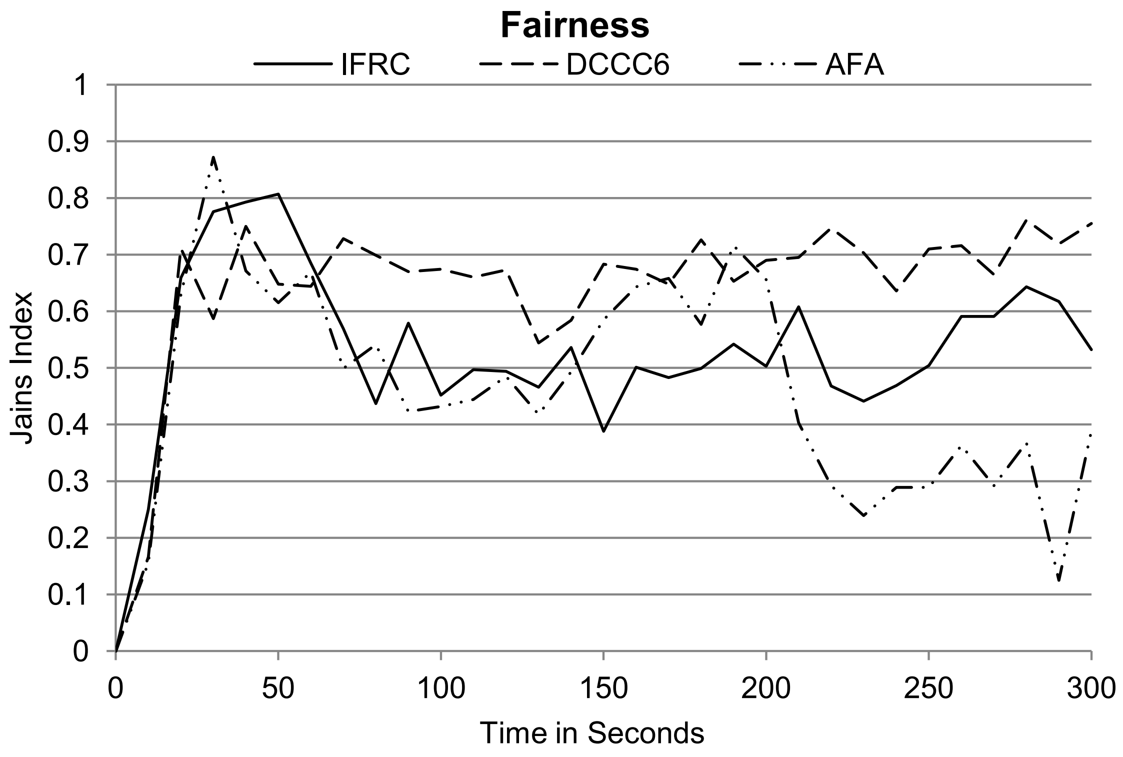

Figure 5.9: Jain's Index fairness over 10 second intervals

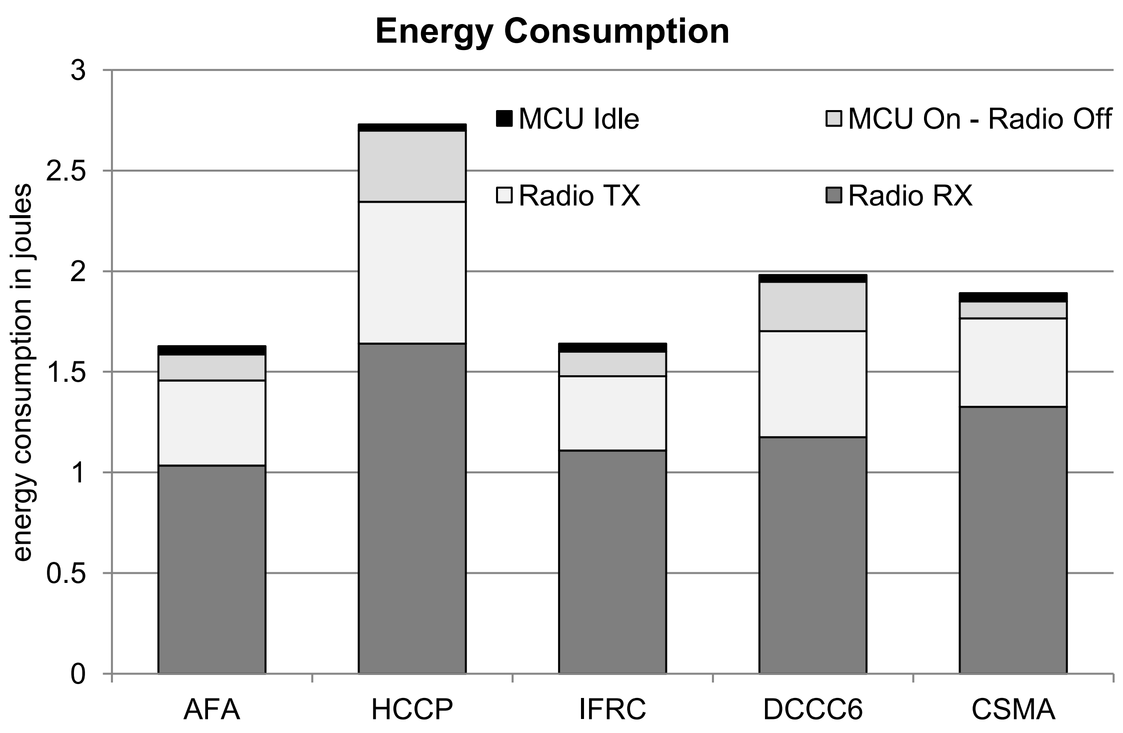

Figure 5.10: Average energy consumption per node *(for the whole duration of the experiment)*

5.5 Testbed Experiments

5.5.1 Testbed Configurations

Subsequently, we investigated the performance of DCCC6 and CSMA MAC in a 15 node multihop indoor testbed *(Appendix D explains in detail why we picked 15 nodes for our test-bed)*. Our deployment is based on Sensinode N740 Nano-Sensors (TI/Chipcon cc243x System-on-Chip with IEEE 802.15.4 low power RF transceiver, 8KB volatile RAM and 128 KB Flash). For this work we used our port of the Contiki OS for Sensinode /cc2430 devices [145]. The radio at the nodes was configured with an output power of -25.2 dBm. The firmware on each device implements a simple UDP server, which waits listening for a packet which signals the start of a test. Upon reception, the device will start transmitting until it has sent 3000 datagrams or until the UDP server receives a “stop” message. Similarly to the simulation experiments the transmission rates used at the source was 6pkts/sec. The methodology used for the packet queue was the same as the one in the simulations. As described in section 3.11, with this method any of the devices can act as a traffic source and we can test multiple different scenarios in the same deployment. By using a USB border router, we can route IPv6 traffic between the 6LoWPAN and the Internet. The receiver of the flows is an Internet host, controlling tests and performing all logging and necessary measurements.

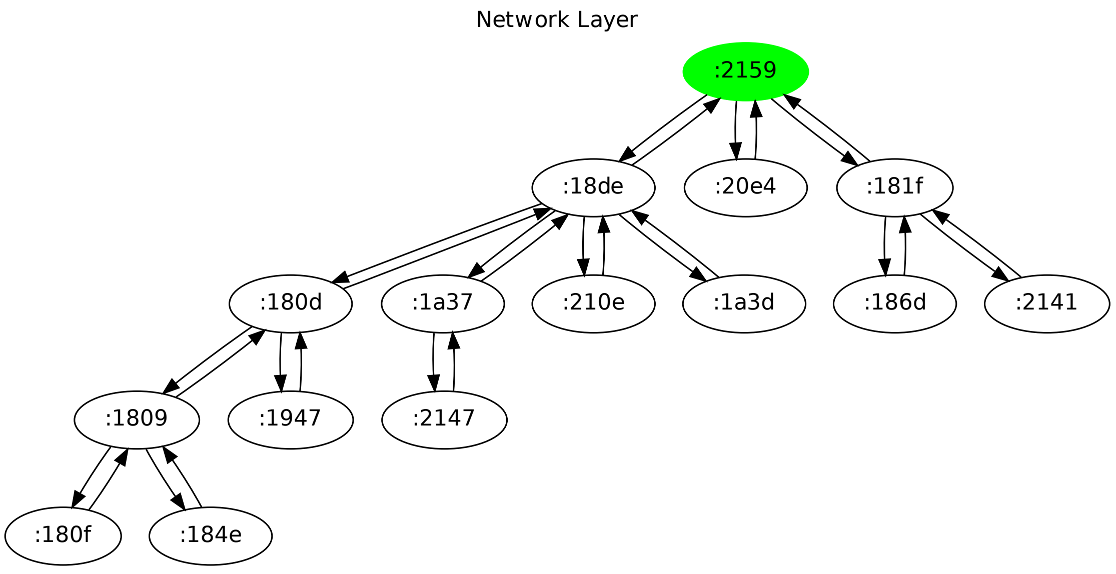

Figure 5.11: Routing tree used in our test bed experiments

After switching the deployment on, the RPL routing protocol converged forming the tree illustrated in Figure 5.11. The number inside each bubble depicts the last two bytes of each node's IPv6 address.

The objective of this testbed was to evaluate the performance of DCCC6 in real hardware and thus during the testbed evaluation, only one source was transmitting at a time. The performance of the nodes was measured for various distances *(in hops)*. For each test iteration, we evaluated two metrics: *i)* goodput as total number of packets received by the UDP server and *ii)* delay as the round trip time from the server to the sensor nodes.

5.5.2 Testbed Results

By examining Figure 5.12, how goodput changes when the number of hops increases can be observed. In this experiment, network density was very high with most of the nodes been in transmission range with each other. The more the distance between the source and the destination, the more times each packet will be transmitted in order to reach the base station. Since nodes are in transmission range with each other, it is understandable that the greater the distance between the source and the destination the more interference between the nodes *(when a node is transmitting most nodes in the tree could receive the packet)*. Taking the above under consideration, we can conclude that the more hops the source is away from the destination, the higher is the probability of collisions, buffer overflows and thus congestion to occur.

Both DCCC6 and CSMA had the same number of successfully received packets at the sink when the distance of the source is 1 hop away. This is expected

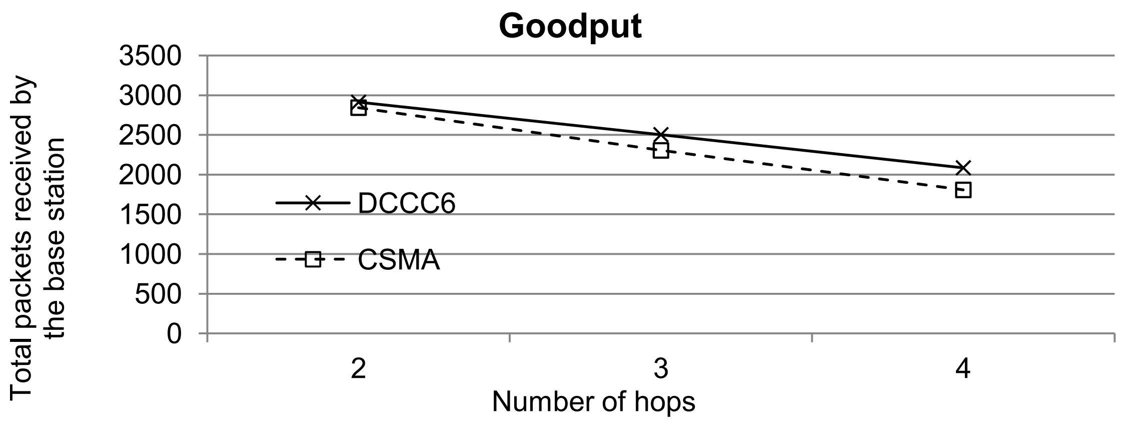

Figure 5.12: Ratio of successfully received packets over the number of hops

since there can't be any congestion when the distance of the source is 1 hop. On the contrary when the distance increased, the difference between the goodput performance of the above protocols became noticeable. It is also worthy to mention, that the greater the distance between the source and the destination the more the gap in their goodput performances grow with DCCC6 having 15% higher goodput when the distance increased to 4 hops.

In our testbed, "ping6" was used for the measurement of the round-trip times. In both heavy and no traffic scenarios one ping6 packet was transmitted every 5 seconds. Figure 5.13 illustrates the round trip delay when our network was idle *(no traffic in the network)* is presented. Since there was no traffic in the network and therefore no congestion, the round trip delay of the compared schemes was similar.

On the other hand, Figure 5.14 shows the round trip delay of DCCC6 and CSMA during congestion. In this figure, it can be observed that during congestion DCCC6 had significantly higher delay. Furthermore, the greater the distance *(in hops)* between the source and the destination the greater the difference between the delay time of the two schemes increased. As mentioned previously in section 5.4, during congestion, with DCCC6 packet queues will be utilised in most of the nodes in a flow path *(congestion mitigation)*. On the other hand with CSMA only the congested nodes will have an increased number of packets in their packet queues. This different behavior can explain why DCCC6 had higher delay times and why the difference in the delay times of the schemes increased as the distance grow.

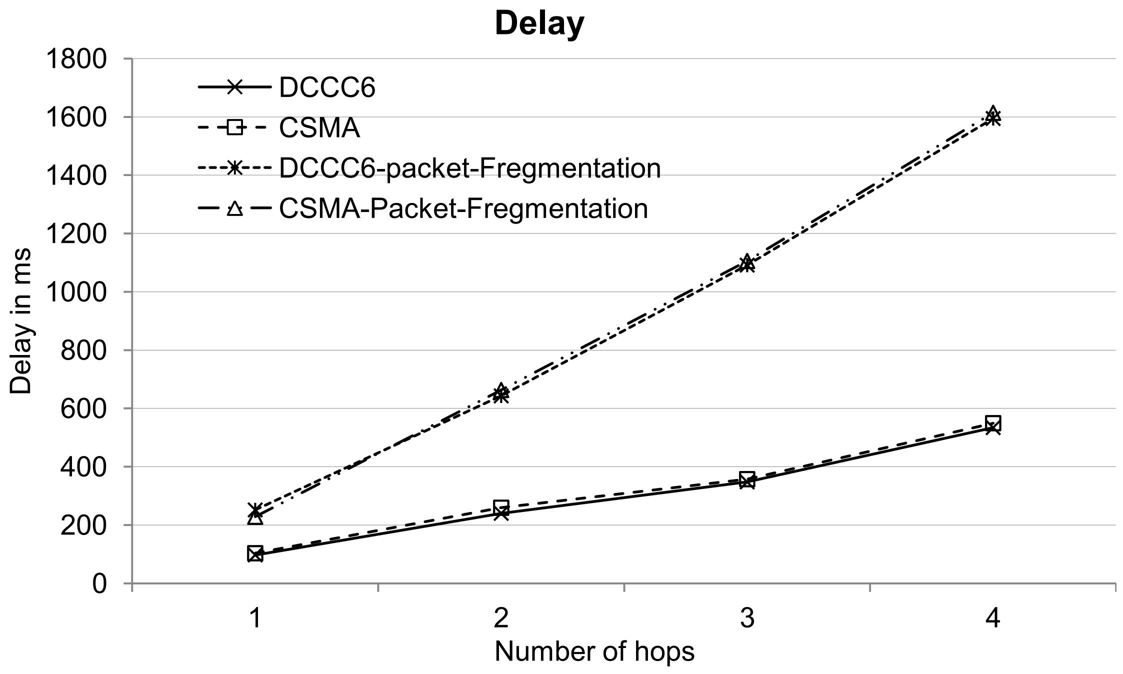

Figure 5.13: Packet delay (round trip)

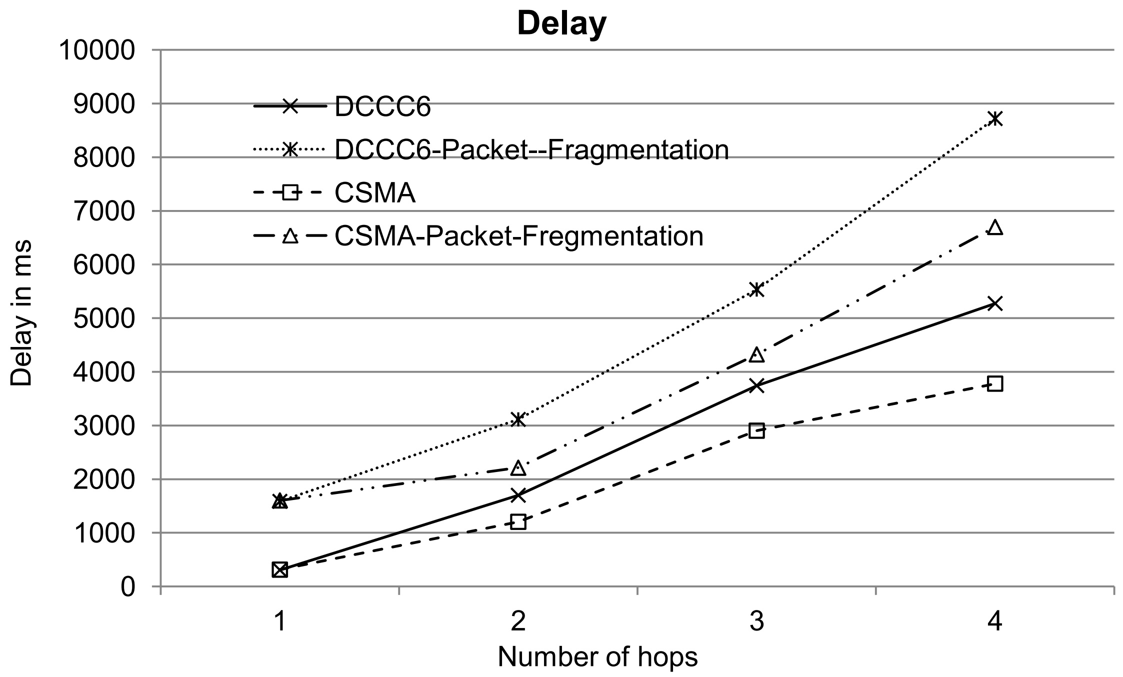

Figure 5.14: Packet delay (round trip)

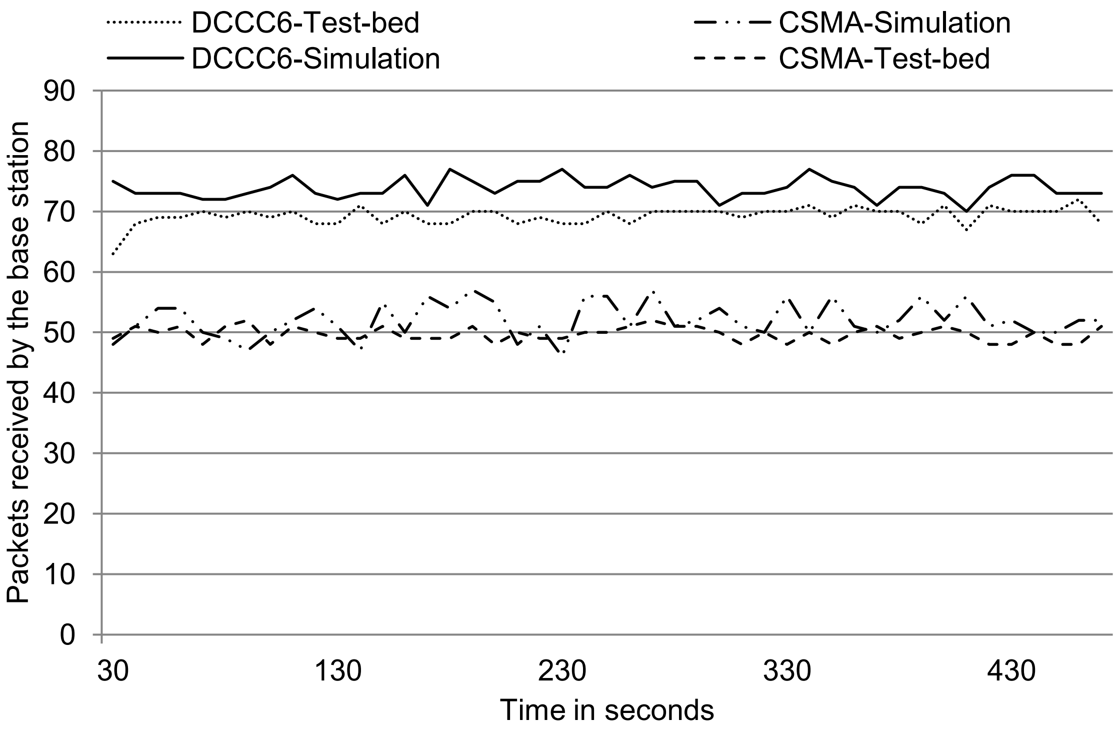

Figure 5.15: Goodput: hardware against simulation

5.6 Simulation Vs Testbed

In section 3.3 COOJA simulator was described. During the simulations, nodes simulating real hardware based on tmotesky platform were used. By simulating nodes emulating the behaviour of real hardware more realistic results can be achieved *(results closer to real test-bed)*. Even though, due to various reasons described in section 3.10, it is very hard to achieve identical network conditions between simulations and test-bed. This in turn can lead to inconsistencies between results achieved from simulations and test-bed.

The test-bed experiments done for DCCC6 and presented in subsection 5.5.2 aimed in validating the performance and functionality of DCCC6 in real hardware. Therefore, even though successfully demonstrated similar behaviour to the simulations they are not as suitable for comparison with the simulation results. During the next chapters of this work section 6.7, simulation and test-bed experiments are going to be compared throughly in as similar as possible environments. That way an understanding of the performance difference and constrains between simulations and test-bed are going to be more visible for every experiment.

Since it is hard to relate the previously conducted test-bed experiments with the simulations, a comparative experiment is made for further evaluation. In order to get an idea of the relation between the simulated and test-bed experiments, network conditions must be as similar as possible. Additionally, the experiment must be under heavy congestion conditions since most network failures and ab-

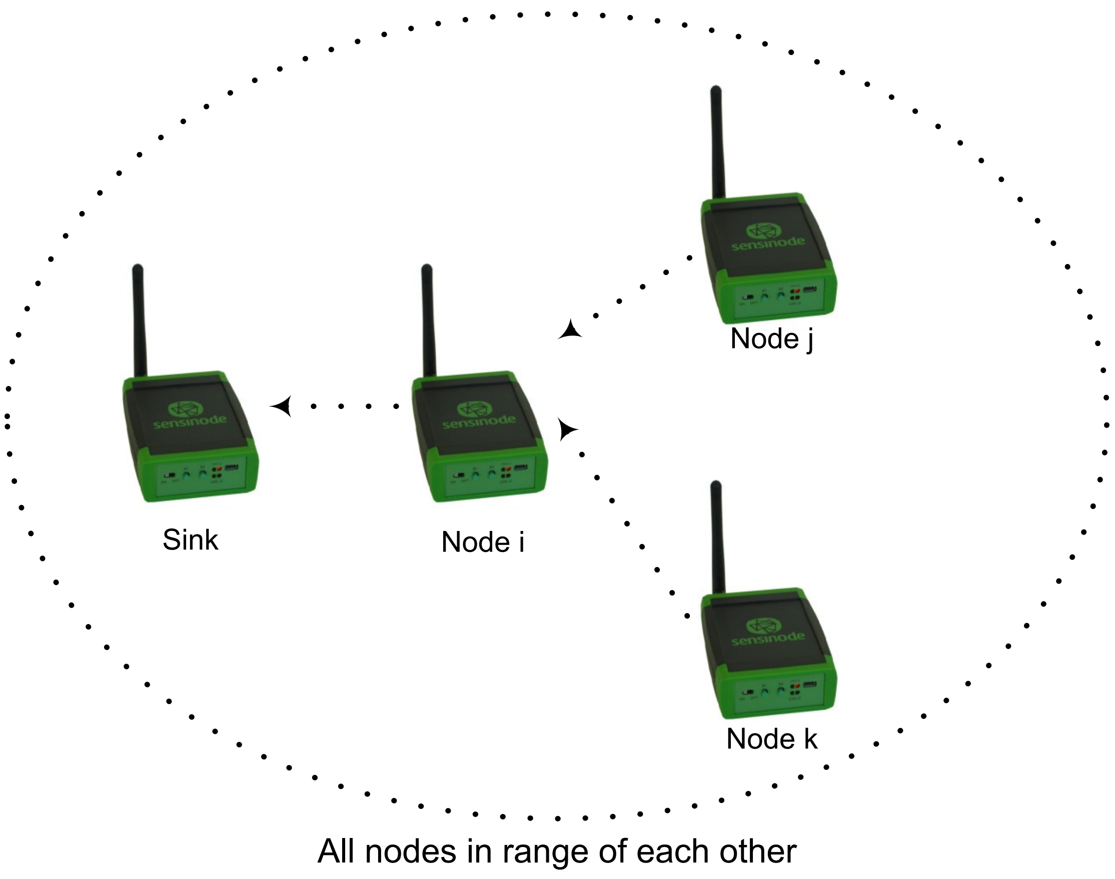

Figure 5.16: Simulation and Test-bed common schenario

normalities this study is focused on occur during this conditions. So that we can satisfy the above requirements a simple four node scenario was used. Figure 5.16 demonstrate the node topology used for the comparative experiment. In this scenario, all nodes were in range of each other in order to increase the network density and thus the collisions occurrence and congestion. Out of the four nodes, two were sources one node was intermediate and one the sink node. The application layer software at the sources was flooding the network with packets by transmitting one packet every 62.5 ms *(this way intense congestion will be achieved in the majority of the experiment)*. The mechanism used for the packet queue was the same as the one used for the previously discussed simulation and test-bed experiments.

The performance of DCCC6 and CSMA MAC, in both simulation and test bed experiments are presented in Figure 5.15. Even though both schemes had similar performances between simulation and test-bed experiments, their performance was noticeably better during the simulation experiments. There are two main reasons behind the better simulation performance of the algorithms:

1. COOJA is simulating Sky motes which are different than the Sensinode used for the test-bed. Even under the same operating system different platforms and thus different hardware can behave slightly different in some occasions

(the most important difference in the hardware wich affects the results will be discussed in the next paragraph).

2. It is very hard if not impossible, to create the same radio conditions in the simulations and the test-bed. In reality the environmental conditions affecting the radio transmissions at the nodes can always change. On the contrary simulators always have a stable, consistent radio environment. The above observation was discussed in detail during section 3.10.

When a collision occurs, CSMA triggers a back off algorithm for future retransmissions of the packet. This back off algorithm, uses a random algorithm for the calculation of the next transmission time. Sky mote random generator takes as a random seed parameters related to the node's ID. Usually, the node ID is created by the MAC address but simulated nodes does not have any hard coded MAC address and thus rime addresses such as: 1 - N, where N is the number of nodes in the simulation, are used. This in turn adds less complexity to the random seed. Additionally, Sky motes are using the MSP430 random generator *(software)* and the software is initialized with a constant seed. This can lead to the generation of the same random numbers *(in some cases)* unless you have manually assigned a node id. On the other hand, the nodes used in our testbed *(Sensinode)* use a hardware random generator. The packet retransmission back off algorithm used in the MAC layer, uses the formula shown below for the retransmission of a packet:

$$R_i = C_i + (Rand_{no} \% Ret_{no}) \tag{5.4}$$

Where R_i is the retransmission interval, C_i is the current inter packet transmission interval, $Rand_{no}$ is a randomly generated number and Ret_{no} is number showing how many times the packet has been retransmitted. C_i is always a fixed and unchanged number in Contiki's CSMA. Therefore, the importance of the random number in this formula is easy to understand.

Taking under consideration the above observations, it can be concluded that the random generation used by the back off algorithm of CSMA had lower randomness during simulations. This lead to more frequent and even consequent collisions which in turn can justify the high jitter observed in the simulation performance of CSMA. CSMA demonstrated a much less jittery behaviour during the test-bed experiments.

A further analysis shows that the performance difference of DCCC6 was much greater than the performance difference in Contiki CSMA. The reason behind this is the CA mechanism used by DCCC6. The congestion avoidance mechanism in

Table 5.5: Size of the various schemes in bytes

	HCCP	AFA	IFRC	DCCC6	CSMA
Simulation (Compiler: MSP430)					
TEXT	1884	1174	1824	1790	730
DATA	102	16	68	78	12
BSS	154	188	150	182	88
Hardware (Compiler: SDCC)					
CODE	5259	3979	4888	4795	2796
XRAM	330	272	301	314	103
CONST	92	121	94	27	26
DATA	0	0	0	0	0
BITS	0	0	0	0	0

DCCC6 assigns different transmission rates to the nodes causing congestion *(C_i)* and thus the estimation of R_i has a greater degree of randomness in DCCC6. This in turn explains both DCCC6's noticeably bigger performance difference and why this scheme didn't demonstrate as high jitter during simulations. In Table 5.5 the sizes of the algorithms used for our experiments are presented. It is notable that MSP430 compiler produced more than two times smaller executables than SDCC compiler.

What parts of the programme are mapped in each of the CODE, XRAM, CONST, DATA and BITS is described in detail in section 3.4. The related specifications for MSPGCC compiler are summarised bellow [146, 147]:

DATA: This portion contains the program's data part. It further divided into

1. Initialised Read Only Data - This contains the data elements that are initialised by the program and they are read only during the execution of the process.

2. Initialised Read Write Data - This contains the data elements that are initialised by the program and will be modified in the course of process execution.

BSS: This contains the elements that are not initialised by the program and are set 0 before the execution of the processes. These can also be modified and referred as BSS(Block Started Symbol).

TEXT: This portion contains the instructions to be executed. On many Operating Systems this is set to read only, so that the process can't modify its instructions. This allows multiple instances of the program to share the single copy of the text.

5.7 Summary and Discussions

In this chapter, the problem of congestion control in 6LoWPAN networks with duty cycling is addressed. DCCC6, a new CC scheme which considers the existence of an underlying duty cycle protocol has also been proposed. DCCC6 performs congestion detection based on a dynamic buffer. When congestion occurs, parent nodes will inform the nodes contributing to congestion and rates will be readjusted based on a rate adaptation scheme. The child notification procedure will be decided by DCCC6 and will be different when the network is duty cycling. When the network is duty cycling the child notification will be made through unicast frames. On the contrary broadcast frames will be used for congestion notification when the network is not duty cycling. Additionally, the main differences between IPv6 and non IP sensor networks as well as how duty cycle algorithms can affect existing CC schemes are discussed.

DCCC6 is developed as a stand alone MAC. Therefore, it is protocol independent and can be used with various WSN applications. Additionally, DCCC6 confronts congestion in a hop-by-hop fashion and thus it's congestion mechanisms are not related to the topology and the routing protocol in the network. Moreover DCCC6 can forward packets based on their priority and successfully detect and confront congestion under both duty cycle and non duty cycle WSN.

Simulation and testbed results show that DCCC6 is better than previous works in terms of goodput and packet loss. Energy consumption, delay and fairness levels were also competitive when compared to other mechanisms with DCCC6 maintaining the overall lowest energy consumption per successfully received packet.

This work is the first work on congestion control for Contiki OS. Furthermore, DCCC6 is one of the first congestion control schemes that consider the existence of duty cycle and it is tailored in the characteristics of IPv6 6LoWPAN networks. Therefore DCCC6 has is of great significance for the Contiki OS and WSN community. In the future, DCCC6 can be used as a paradigm by congestion control protocol architects using both Contiki or other commercial operating systems for WSN. Even though DCCC6 is based in the characteristics of IPv6 and 6LoWPAN networks it is designed as a stand-alone MAC layer and can easily be combined with other protocol stacks.

Chapter 6

CADC: Congestion Aware Duty Cycle MAC

The aims of the chapter is to propose a complete and stand alone MAC designed in Contiki OS based on the arbitrary traffic characteristics of IPv6 WSN.

6.1 Introduction

A wireless sensor deployment usually consists of multiple nodes monitoring their surrounding environment. Collected data traverse the network towards sink nodes in a multi-hop fashion. With the growing maturity of wireless sensor networks *(WSN)* the attractiveness of their flexible deployment scheme enters the focus of applications from a variety of domains. The majority of WSN implementations require large numbers of battery powered devices, capable to autonomously operate and communicate. The flexibility of these networks comes however at the price of high demands concerning the implementation of protocol stack. Recently, IPv6 over Low power Wireless Personal Area Networks *6LoWPAN* and related specifications [148, 149] are becoming increasingly popular in the WSN community. Those technologies promise better scalability in terms of number of nodes, due to the extended IPv6 addressing scheme. Additionally, the integration between 6LoWPAN and the Internet now becomes a question of network layer routing and alleviates the need for application layer gateways [5].

In order to achieve long network lifetime, energy-efficiency must be a primary concern in WSN deployments. Data aggregation and energy-aware routing are some of the primary energy saving techniques. It has also been shown that idle radio listening is one of the most major sources of energy consumption [6]. In order to prolong sensor deployment life cycle, research community members have made significant efforts in the development of Radio Duty Cycling *(RDC)* Medium

Access Control protocols *(MAC)*. With radio duty cycling, nodes turn off their transceivers for long periods of time in order to minimise idle listening. Crucially, as demonstrated in chapter 4 and chapter 5 the unique characteristics of duty cycle algorithms and 6LoWPAN sensor networks can have significant impact on the performance of a WSN deployment.

In sensor applications, a burst of traffic can be caused by a sudden event, resulting in network congestion. Under these conditions, MAC protocols with fixed duty cycle will suffer from data loss due to their inability to adapt to traffic needs. In order to avoid packet loss, the fixed duty cycle [7, 8, 9, 4] should be increased. However, increased cycles will lead to longer periods of radio listening, which in turn will lead to higher energy consumption. Hence duty cycles should only increase under heavy load and decrease when the network is idle.

Recently, there were significant efforts towards the development of dynamic duty cycle MAC protocols [10, 11, 12, 13, 14, 15, 16, 17]. Conceptually, a number of MAC protocols with capabilities of traffic-based cycle adaptation have been proposed. Previous works, have not carefully considered the unique characteristics of IPv6 WSN and it's arbitrary traffic patterns in their design.

In this section, **CADC:** a congestion-aware duty cycling MAC protocol is proposed. The proposed MAC can operate with any routing topology, traffic patterns and it is protocol independent. Our congestion aware MAC prevents packet loss by increasing the duty cycle under heavy traffic and saves energy by decreasing the duty cycle in light traffic. CADC is evaluated and compared with other dynamic and fixed duty cycle protocols in both COJA simulator and real hardware test-bed. The scheme is extensively tested under various traffic patterns *(different source rates, distances in hops and topologies)* and we demonstrate that it exhibits an overall better performance than both fixed duty cycle schemes and other dynamic duty cycle protocols in terms of energy consumption, goodput, delay and packet loss.

The rest of the section is organised as follows: a discussion on the limitations of dynamic duty cycle protocols can be found in section 6.2. Section 6.3 discloses CADC's design details. The simulation experiments and analysis thereof can be found in section 6.5 while tes-bed experiments will be analysed in section 6.6. A comparison between simulation and test-bed results will be presented in section 6.7. Lastly, section 6.8 summarises our main results and describes future research directions.

6.2 CADC Design Considerations

Contrary to traditional, application-centric WSN design, a 6LoWPAN may be the host of a variety of applications (e.g. network management, data collection, CoAP: Constrained Application Protocol services) simultaneously. In this case:

1. packets from different applications will have different sizes, priorities and possibly different destinations
2. Traffic may be bi-directional
3. Applications may use different protocols for communication (UDP or TCP)
4. Large packets may get fragmented by the 6LoWPAN adaptation layer.

Considering the above characteristics, a dynamic duty cycle protocol for 6LoWPAN sensor networks should:

1. Be protocol-independent,
2. Be topology-independent,
3. Forward packets based on their priority,
4. Implement a traffic-aware method for the calculation of the required duty cycle, with nodes along the traffic path adjusting their cycles in a uniform fashion.

As discussed in chapter 5 and section 3.6, duty cycling nodes switch their radio transceivers on and off for energy saving. Usually, nodes will also turn their radio off when collisions occur [4]. As the network density increases, so does collision probability. When a collision occurs, a back off time for packet retransmission will be calculated randomly. Even with the most advanced synchronisation algorithms, the existence of collisions render perfect synchronisation between nodes very challenging. In order to address it, many existing protocols transmit link layer frames repeatedly *(multiple strobes)* until they receive a successful acknowledgment, or until a collision is detected. This approach, which is also known as a 'packet train', works efficiently for link layer unicasts, however broadcast packets never get acknowledged and thus are inefficient [4]. Therefore, a dynamic duty cycle protocol must not rely on broadcasts for information sharing between the nodes.

Under dynamic adjustment schemes, calculation of the new duty cycle can be:

1. centralised, whereby a single node calculates the new DC and informs the remainder of the network,

2. distributed, where each node calculates its own optimal DC based on information available locally. With the first approach, problems such as faulty connection links due to different cycles are prevented since all nodes will update to the same DC. This in turn will lead to unnecessarily high energy consumption at idle nodes and longer adaptation times.

Distributed protocols can adapt to traffic requirements faster, since each node can change its DC without waiting for the entire network to converge to the same cycle. This is also more energy efficient, since only active nodes will adjust their cycles. On the contrary, this approach may result in incorrect cycle synchronisation between the nodes. If a node increases its cycle and its parent node fails to receive that information correctly, the communication link between the two nodes will become unreliable. Therefore, designing a distributed scheme can be a very challenging task in an IPv6 and 6LoWPAN WSN where the traffic patterns are arbitrary. Usually, distributed duty cycle protocols, use periodical wake-ups in order to listen for packet transmissions from neighbour nodes. If a packet transmission is detected during a wake-up, the receiver keeps the radio on to receive the packet. When a packet is received, the receiver sends a radio acknowledgment. To transmit a packet, sender repeatedly sends the packet until the reception of a link layer acknowledgment from the receiver. Broadcast packets does not wait for link layer acknowledgments, with the transmitter instead continuously sending the packet for the whole wake-up interval [4]. Therefore, in the majority of dynamic duty cycle protocols, the duration of each packet's transmission is related with the node's wake-up frequency. When the above packet transmission approach is used by a dynamic duty cycle protocol; there is a possibility for incorrect cycle adaptations to occur. How this incorrect cycle adaptations can occur can be explained if we take under consideration the following example:

Lets assume that node i is the transmitter and node j the receiver. If i decreases the interval between its radio wake-ups *(higher cycle)* and j does not, i's continuous packet transmissions will not be long enough to be detected during j's wake up periods. Based on that, it is easy to conclude that node i will transmit packets while node j is still in sleep mode. Consequent failure of packet transmissions may lead to a new, cycle adaptation in node i which in turn may result in a worse situation such as path adaptation *(a node will change its parent node)*. Thus dynamic duty cycle protocols with distributed cycle adaptations must be designed carefully and successfully share the duty cycle information between nodes.

All of the above considerations, have been taken into account in the design of the proposed protocol, and thus CADC is tailored on the versatile needs of an IPv6 WSN.

6.3 Implementation of CADC

The design of the proposed *CADC* is focused on overcoming all existing challenges related to DC in 6LoWPANs, such as the ones discussed in section 6.2. Furthermore, *CADC* is routing, application and traffic independent. It operates efficiently under varied traffic patterns and with multiple applications operating simultaneously. Even though CADC is designed for 6LoWPANs, it can easily be applied to any WSN deployment.

In this research study, the radio duty cycle of a WSN is expressed as a function of node wake-up frequency called channel check rate *(CCR)*. More details about the relation of channel check rate and radio duty cycle can be found in [4] and section 3.6.

6.3.1 General Characteristics and Rules

In order to achieve fast adaptation times, CADC measures congestion levels based on node activity. Congestion level sampling is not performed when nodes are idle, since congestion can not occur. On the other hand, when traffic increase so does the frequency of congestion level sampling. In order to achieve this, every n_i packet transmissions each node will measure its queue occupancy and packet transmission ratios *(successful, not acknowledged, collisions)*. If a node has no in-traffic at the MAC layer *(can be both self and from other node traffic)* within a period of time P_t; it is considered idle and thus congestion levels are not measured. Additionally, a node will return its channel check rate back to the default if it becomes idle. The period of time P_t, is proportional to the node's channel check rate *(higher channel check rates have smaller periods since they can transmit a greater number of packets per second)*. Moreover, two queue thresholds Q_h and Q_l are implemented at each CADC node. Nodes are considered to be in a light congested state if queue occupancy exceeds the low queue threshold Q_l and in heavy congestion state when queue occupancy exceeds the high threshold Q_h. A transmission threshold U_T is applied to the packet transmission statistics *(measured during congestion sampling)*, in order to provide more efficient congestion detection. Each node will also measure its incoming traffic *(in packets)*. The aforementioned metrics are then combined for the reconfiguration of the channel check rates at the nodes.

Overall, CADC's dynamic CCR adaptation is designed to achieve energy efficiency, high performance, scalability and very fast adapting times without need for synchronisation between nodes.

Even though CADC is designed so as nodes can operate independently *(no time synchronisation between nodes)*, every CADC node must follow some rules:

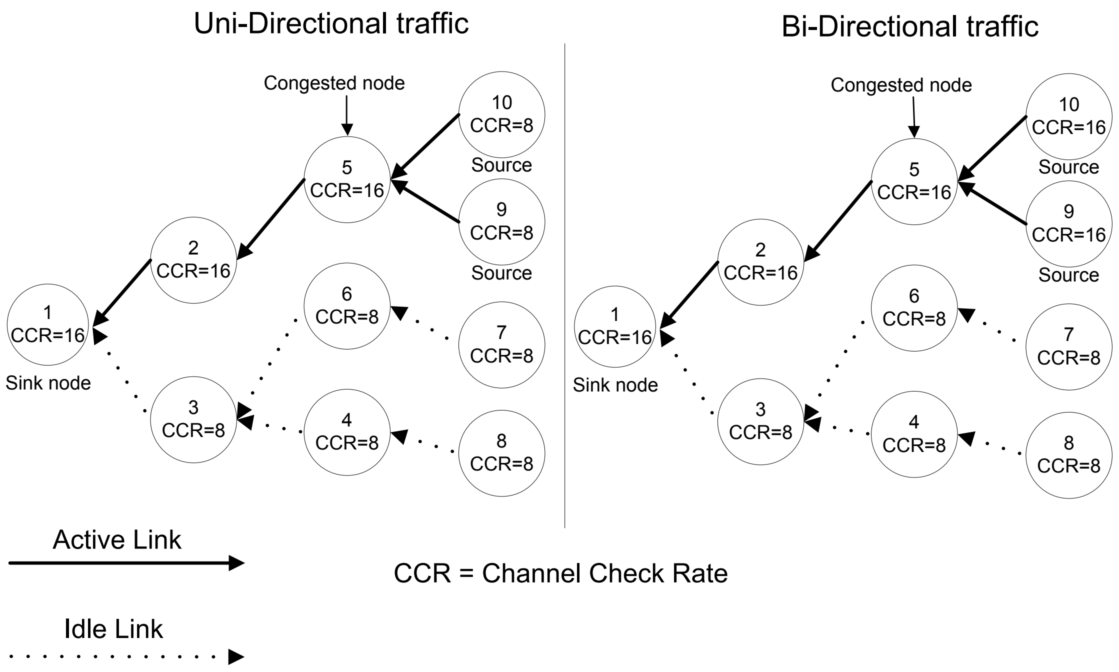

Figure 6.1: CADC Channel check rate adaptation scenarios for different traffic patterns

1. A CADC node must always have the same or lower channel check rate than its parent nodes *(in bi-directional traffic, a node must have the same DC as its parent and children)*. A detailed explanation of how CADC nodes will adjust their channel check rates in different traffic patterns, can be seen in Figure 6.1.

2. Information about a node's channel check rate must frequently be forwarded to parent nodes.

3. When a node becomes idle, its channel check rate will return to the default configuration *(minimum channel check rate)*.

In order to satisfy the above rules and maximise the performance, CADC proposes a 5-state operation mode at each node. Nodes will measure their congestion levels and enter a state accordingly. The 5 congestion states are presented in detail in the next section.

6.3.2 Measuring Congestion Levels and Cycle Adaptation

Various techniques have been proposed in the wireless sensor literature for the measurement of node congestion. In recent studies [47] and through the experiments conducted in chapter 4, it is reported that queue occupancy is sufficient when traffic patterns are many-to-one, which is the predominant case for 6LoWPANs. In case of arbitrary communication traffic patterns with varied duty cycles between the nodes, measuring congestion based on queue length may be insufficient. Nodes require a higher degree of knowledge in order to efficiently adjust their duty cycles. For the successful calculation of the optimal channel check rate in each node, CADC is based on five parameters:

1. The ratio of successful transmissions *(in %)* over the transmission threshold:

$$R_{succ} = \frac{T_{succ}}{U_T} \tag{6.1}$$

2. The ratio of failed transmissions *(in %)* over the transmission threshold:

$$R_{fail} = \frac{T_{fail}}{U_T} \tag{6.2}$$

3. The packet-queue occupancy

4. The ratio of successful transmissions over failed transmissions:

$$R_{service} = \frac{T_{succ}}{T_{fail}} \tag{6.3}$$

   If $T_{fail} = 0$, $R_{service}$ is considered > 1without running the above calculation.

5. The highest channel check rate announced by a node's children ($CCR_{announced}$).

Every n_i transmitted packets, these parameters are recalculated *(n_i must be proportional to the maximum queue size in order to achieve minimum reaction times)*. Upon the calculation of the above parameters, CADC's duty cycle adaptation algorithm will be executed and the node will enter a state. A detailed transition diagram of CADC's states can be seen in Fig. 6.2.

The five CADC states are:

- **Congestion collapse state:** This state signifies that the node is heavily congested and the channel check rate must significantly be increased. A node will enter this state if:

  1. its packet queue levels are above the high threshold Q_h. This indicates that it is likely packets to be dropped in the near future. Therefore we need to significantly increase the cycle to prevent it.
  2. R_{succ} is below 1. This indicates that the queue keep increasing fast because the node has a very high percentage of failed transmissions.

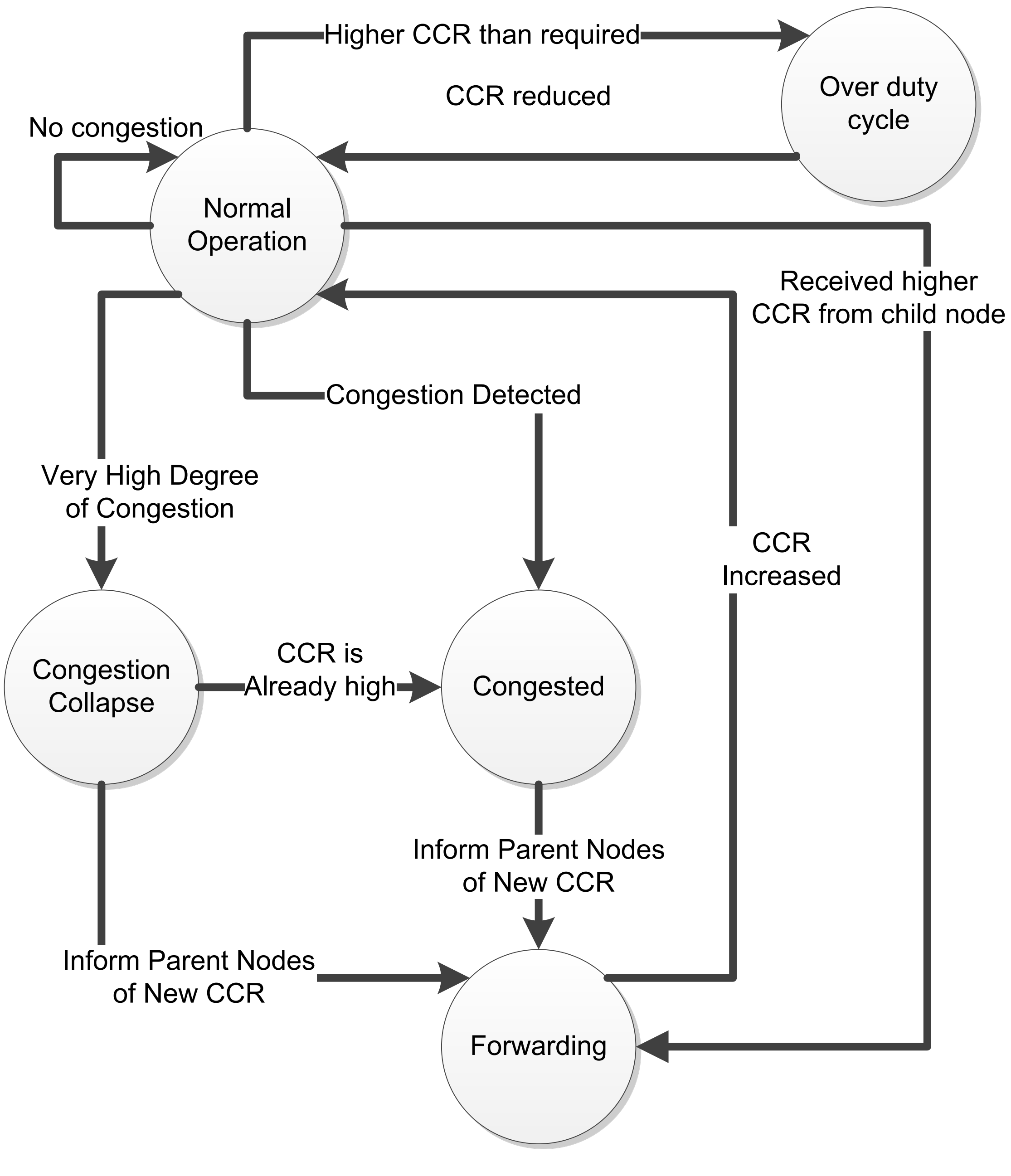

Figure 6.2: State transition diagram of CADC's 5 operating states

3. finally that nodes channel check rate must be significantly lower than the highest channel check rate max_{CCR} *(the highest value a channel check rate can take must be defined in advance).* We can't increase the cycle more than the maximum configuration.

If the above parameters are met, a node is highly likely to drop packets and thus there is a need for an immediate duty cycle increase. When a node enters this state its channel check rate will be quadrupled.

- **Congested state:** When a node is not heavily congested, but predicts that there is a possibility of congestion occurring during the next packet transmissions it will enter this state. In this state, a minimum will be made to the channel check rate. Therefore the risk of over-increasing channel check rate and thus have unnecessary energy consumption will be decreased. More specifically, a node will enter this state if:

  1. its packet queue level is above the lower threshold Q_l. This indicates that the node is congested but packets are not likely to be dropped yet.
  2. $R_{service}$ is below or equal to 1. This indicates that the queue is not draining and it will keep growing.
  3. Nodes that entered the congestion collapse state, do not operate in the maximum channel check rate but cannot quadruple their channel check rate will also enter this state.

- **Over duty cycle state:** This state represents that the channel check rate in the node is higher than the required. A node will enter this state when:

  1. Packet queue level is below the lower threshold Q_l.
  2. the channel check rate in the node must be higher than $CCR_{announced}$. Nodes cannot disobey the rules *rule - 1*
  3. $R_{service}$ must be equal or greater than 1 and R_{fail} must be below 1. This way nodes can confirm that the queues are draining and there is no need to operate in a higher than the required cycle.
  4. the node's channel check rate must be higher than the minimum channel check rate min_{CCR}. Nodes cannot operate in lower than the minimum cycle.

  When a node successfully enters this state, its channel check rate will be halved.

- **Normal operation state:** This is the default operation state of CADC protocol. Nodes in this state will maintain their current channel check rate configuration. A node will remain in this state if no congestion is detected *(either node is idle or traffic is not high enough to cause congestion).*

- **Forwarding state:** This is an intermediate state with the purpose of informing receiver nodes about future changes of the transmitter's channel check rate. After a node enters congestion collapse state or congestion state, a new channel check rate will be calculated for that node. Before a node adjusts its channel check rate to the calculated value, it will enter the forwarding state and remain in this state for a duration of time *(guard period)* related to the minimum channel check rate *(min_{CCR})*. During this period, nodes will propagate their new channel check rate to any possible receive nodes. Moreover, relating guard period to min_{CCR} will result in having nodes with higher channel check rates remain in this state for a greater number of packet transmissions than nodes with lower channel check rates. This is important since nodes with lower channel check rates are more likely to have a packet received in a single transmission *(as discussed in section 6.2, the duration of each packets continuous transmission is higher for lower channel check rates).* When guard period finishes, nodes will adjust their channel check rate to the calculated value.

In order to explain in detail the role of *Forwarding state* in CADC, lets assume that node i is the transmitter node and node j a receiver. When node i detects congestion, it will enter congestion collapse or congested state and the optimal channel check rate will be calculated accordingly. Before node i adapts its cycle, node j must be informed about the change *(rule 2)*. In order to achieve that, node i will enter the forwarding state and insert the information about the future change in its channel check rate to every out going packet. When node j receives a packet from i it will extract the channel check rate information from the packet, enter forwarding state and further propagate this information.

In addition to the mentioned states, when a CADC node becomes inactive *(no transmitted packets for a time period)*, it will set its channel check rate to min_{CCR}. This functionality allow nodes to independently return their cycles to the default configuration.

6.3.3 Duty Cycle and Information Sharing

In order to achieve optimal network performance, it is necessary for each node to propagate its current channel check rate to its parent nodes. For the propagation of channel check rate information, CADC uses the 3 reserved bits of the FCF field in the 802.15.4 frame header (Figure 6.4). Before each unicast transmission, nodes will set a flag in the 3 bit field. Through that flag, a node can calculate the channel check rate at its child nodes. If the channel check rate received is higher than the receiver node's current channel check rate, the node will enter the forwarding state and further propagate this information. When the node exits the forwarding state, it will adjust its channel check rate to the same value as the one received by its child node. Moreover, a node can not reduce its channel check rate to a lower value than the highest received.

A scenario of how CADC will adjust the nodes channel check rate during uni-directional traffic can be seen in Figure 6.3. In the majority of cases, the information will successfully be shared between the nodes through the forwarding state. But in IPv6 WSN, a node may start transmitting or change traffic patterns at any time *(multiple applications running in the same node, some with one-way traffic and others with bi-directional traffic).* Taking this under consideration, CADC must incorporate some intelligent mechanisms capable of detecting traffic patterns changes and adjusting the channel check rates accordingly. These mechanisms will be presented in detail at the next Section.

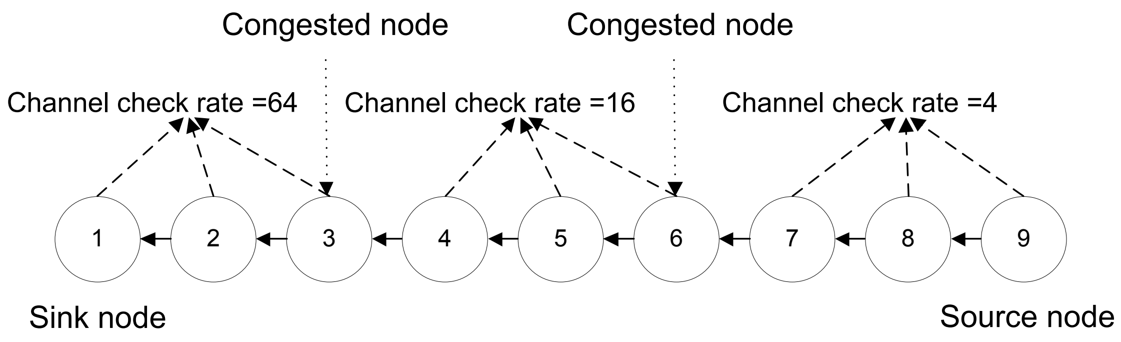

Figure 6.3: CADC traffic flow operation scenario

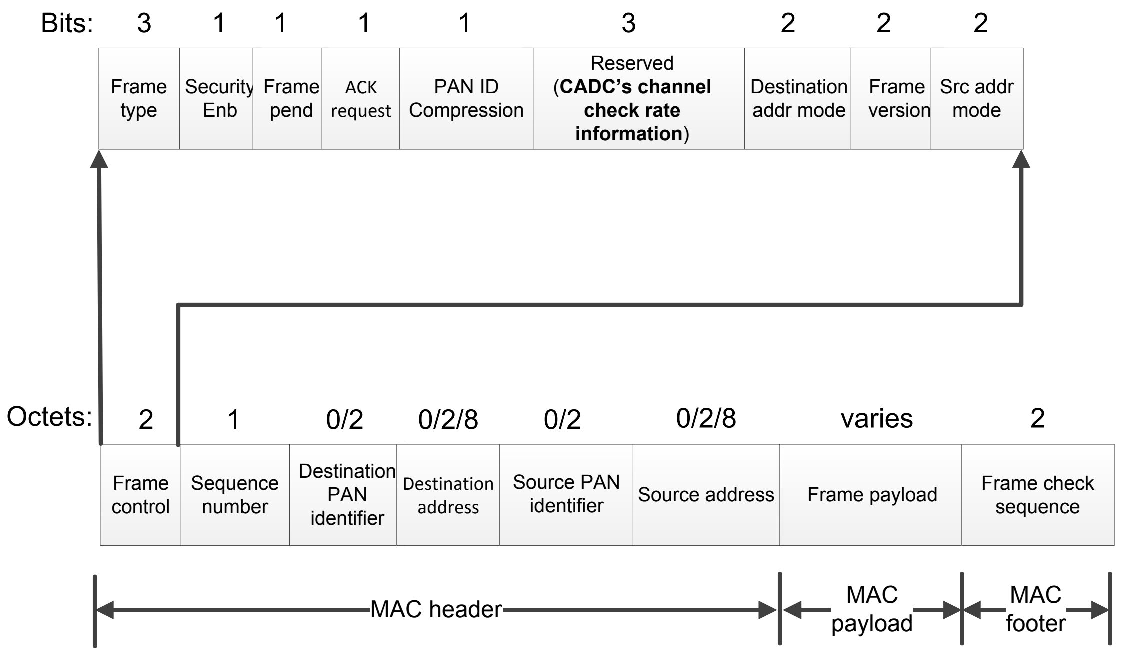

Figure 6.4: 802.15.4 frame header and the 2 bytes of frame control field (FCF)

6.3.4 CADC Packet Retransmission Scheme

Traditionally, nodes perform tasks such as neighbour discovery with the use of broadcast packets. As mentioned in 6.2 duty cycle protocols such as ContikiMAC will repeatedly transmit a packet for a period of time. The duration of a packet train must be slightly longer than the intended recipient's wake up period. With this scheme there is no need for clock synchronisation between nodes, since the destination will definitely wake up once during the packet train. Since channel check rate *(CCR)* represents the radio wake-up frequency, both packet train duration and node sleep times are calculated based on CCR.

If nodes operate with different channel check rates, information exchanges with broadcast frames will become unreliable since nodes with a lower CCR will not receive the majority of broadcast frames transmitted by nodes with higher channel check rates. In order to avoid broadcast frame losses due to this 'disagreement', CADC sets the duration of broadcast packet trains to a fixed value regardless of CCR.

Fixed transmission times for broadcasts can in turn cause continuous collisions to nodes with high channel check rates since unicast transmission times will be much lower. To avoid this, we improved the traditional CSMA backoff algorithm by applying multiple packet retransmission windows for the different cases of collisions. If a collision is detected during the transmission by a unicast frame, the back off algorithm will estimate a retransmission time based on the node's current channel check rate *(different channel check rates result in different backoff times).*

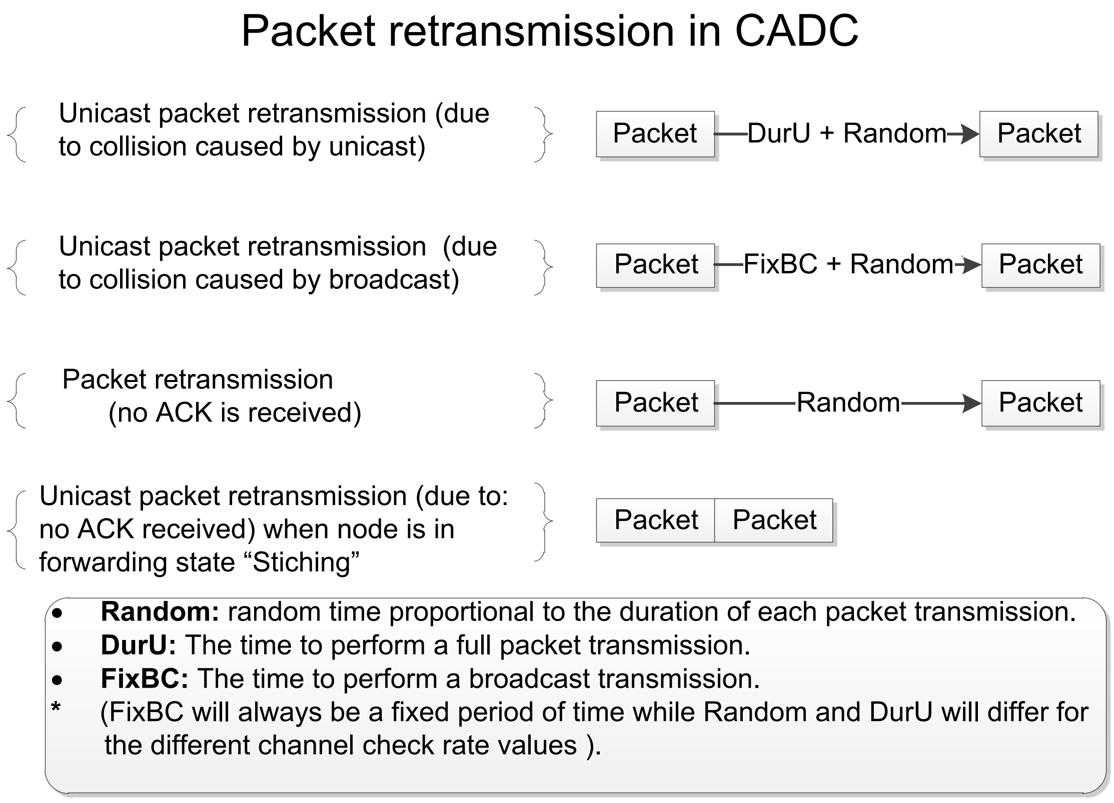
Packet retransmission in CADC
Unicast packet retransmission (due to collision caused by unicast)
Packet
DurU + Random
Packet
Unicast packet retransmission (due to collision caused by broadcast)
Packet
FixBC + Random
Packet
Packet retransmission (no ACK is received)
Packet
Random
Packet
Unicast packet retransmission (due to: no ACK received) when node is in forwarding state "Stiching"
Packet
Packet
Random: random time proportional to the duration of each packet transmission.
DurU: The time to perform a full packet transmission.
FixBC: The time to perform a broadcast transmission.
(FixBC will always be a fixed period of time while Random and DurU will differ for the different channel check rate values).

Figure 6.5: CADC packet retransmission scheme

On the contrary, when a collision is caused by an incoming broadcast frame, the backoff time will be based on the fixed duration of broadcast packet trains. This is illustrated in Figure 6.5.

Traditionally, there are three main reasons why a node will not receive an acknowledgment after a packet transmission: i) the presence of a hidden terminal, ii) data corruption and iii) loss of the link to the destiantion. In case of a dynamic DC protocol with varied cycles between the nodes, it is possible that incorrect DC adaptations may lead to the same problem. To further explain this let's assume that node i is the source node, node j its parent node and node k is the sink node. Now let's assume that node j becomes congested and adapts its channel check rate to a higher value. This will result in node k updating its channel check rate to the same or higher value. At this point, if another traffic flow from node k towards node i starts, the majority of the packets will be lost in the link between node j and node i since node j already operates in a higher channel check rate than node i. This phenomenon will cause a stream of failed, unacknowledged packets. In order to confront scenarios like the one described above CADC nodes will:

1. when a node enters the forwarding state, it will transmit packets based on the maximum transmission time that corresponds to $min_C CR$. This will ensure that the information about the new channel check rate will be propagated even between nodes with different channel check rates. This is very useful in

scenarios where the traffic converges from semi-directional to bi-directional after nodes have adjusted their channel check rates.

2. when a node is in forwarding state and a packet does not receive an acknowledgment, the node will immediately retransmit *"stitch"* the packet. This will ensure that nodes in forwarding state will *even by force* propagate the information to the neighbour nodes.

6.4 Evaluation of CADC

CADC is designed to achieve energy efficiency, high performance, scalability and very fast adapting times without need for synchronisation between nodes. Additionally, how CADC is designed to operate and confront congestion is described in detail during the previous sections .

In this section, two basic experiments *(using 3 nodes)* have been conducted in COOJA and analysed in detail in order to investigate whether CADC can successfully achieve its design targets. Simulations aimed at evaluating behaviour of the protocols in both heavily congested and normal network conditions. During these experiments, the source was configured with CBR traffic and attempted to transmit 200 packets. Detailed simulation logs of the above mentioned experiments can be found in Appendix G, while the detailed performance of CADC during the simulation experiment with low congestion is illustrated in detail by Figure 6.6 and Figure 6.7 *(Text in red represents congestion detection and changes in the nodes status due to congestion while green text represents changes due to node over duty cycling. Text in black colour represents no changes at the nodes status).* It is also worthy to mention that the codes used for the simulation experiments in this section are identical with the ones used for the test-bed experiment *(Contiki can simulate hardware nodes loaded with the actual firmware used for test-bed experiments section 3.3).*

Studying Figure 6.6, it can be observed that for <80 packet transmissions no congestion was detected by CADC. During the transmission of the 80th packet by the source, node's 2 *(intermediate node)* queue occupancy exceeded the Q_l but remained below Q_h. Additionally, node's 2 R_{serv} was below 1 and thus the queue will keep building up. In order to prevent its queue from overflowing, node 2 entered congestion state and updated its CCR to 16 and encapsulated the information on it's outgoing packets according to rule 2. Consequently, node 1 updated its CCR according to rule 1. Even though node's 2 packet queue was higher than Q_l for the packet transmissions between 80 and 90; Node 2 did not detect any congestion and remained in normal operation state. The reason behind

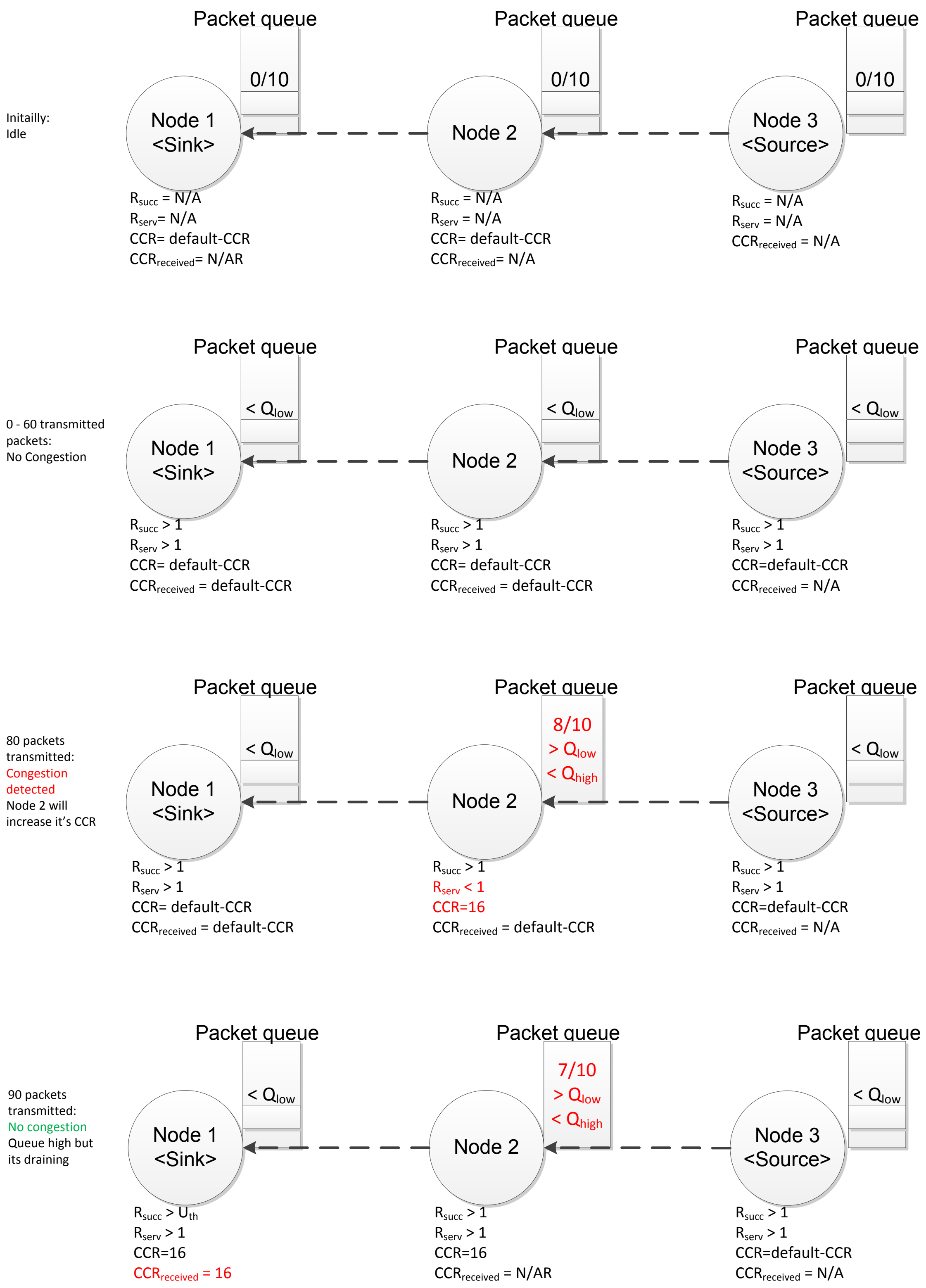

Figure 6.6: CADC evaluation experiment with normal traffic

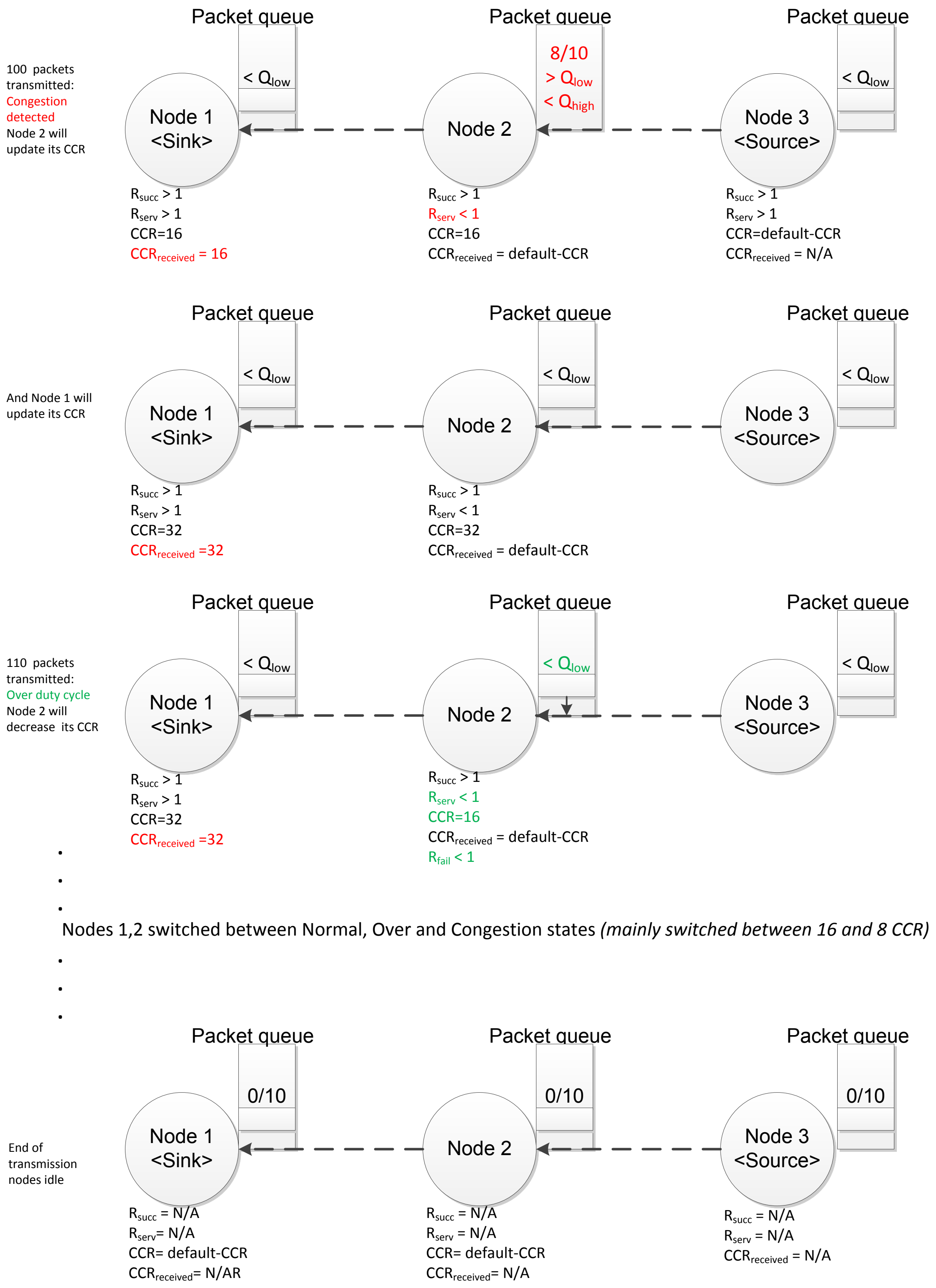

Figure 6.7: CADC evaluation experiments with normal traffic *(part-2)*

this is node's 2 R_{serv} was higher than 1 and thus the queue was draining. Therefore node 2 waited for the queue to drain instead of triggering the congestion avoidance mechanism again. Nodes remained in normal state *(queues drained below Q_l)* and operated without any congestion for another 10 packet transmissions. During the 100th packet transmission from the source, node's 2 queue built up and exceeded the lower threshold Q_l. Moreover, node's 2 R_{serv} was below 1 and thus the node entered congestion state and further propagated the information to its parent node. Both nodes 2 and 1 updated their CCR. 10 packet transmissions later, node's 2 queue had drained and the node had $R_{serv} > 1$, $R_{fail} < 1$ and a higher than the default CCR. Therefore node 2 entered the over duty cycle state and reduced its CCR. From this point on, nodes continued to operate without congestion till the end of the experiment where inactivity was detected and all nodes reconfigured their CCR back to the default.

There was no packet loss during this experiment. On the contrary, during high congestion there was more changes between CADC's states *(detailed logs in)*. Even though, CADC demonstrated similar results in both low and high congestion experiments.

Through these experiments, CADC met all design targets and demonstrated that it can fast adapt to various changes, operate under various traffic requirements and successfully confront congestion while it keeps the CCR at the nodes to the minimum required.

6.5 Simulation Experiments

6.5.1 Simulation Configurations

Through simulations, this section investigates the performance of various duty cycle algorithms and compares them with the proposed CADC. Simulations aimed at evaluating behaviour of the protocols in both heavily congested and normal network conditions. In order to achieve this, over 10,000 simulation runs were conducted: 160 configuration and traffic rate permutations for each protocol in each topology. Each source attempted to transmit 500 packets. The packet transmission intervals *(the time between two consequent packet transmissions)* that the sources have been configured for the different simulation runs was: 500ms, 250ms, 125ms and 62.5 ms and each packet had a 24-byte application data. The tool used to perform the simulations was COOJA [99]. All features of CADC described in section 6.3 have been implemented as a new MAC layer in the Contiki embedded operating system.

The packet queue in every MAC layer used for the simulations *(CADC, BEAM,*

Table 6.1: Different simulation configuration permutations

	ContikiMAC	CADC	X-MAC	BEAM
MAC layer	CSMA	CADC	CSMA	BEAM
RDC layer	ContikiMAC	CADC	X-MAC	BEAM
Channel check rate *(Hz)*	4, 8, 16, 32, 64	*dynamic*	*dynamic*	4, 8, 16, 32, 64
min_{CCR} *(Hz)*	configuration dependent	4	configuration dependent	4
max_{CCR} *(Hz)*	configuration dependent	64	configuration dependent	64
n_i	N/A	10	N/A	N/A
Q_h	N/A	90%	N/A	100%
Q_l	N/A	60%	N/A	60%
U_T	N/A	20%	N/A	N/A

CSMA) was configured to buffer up to 10 outgoing packets *(10 × 128 bytes)*. There was no queue implemented for incoming traffic.

The BEAM protocol was implemented, accordingly to [17] for both MAC and the RDC layer. In Contiki OS and hence our simulations, the RDC is expressed as a function of the wake-up frequency. In our results and graphs this frequency is called channel check rate. For the majority of the platforms in Contiki OS the default value for the channel check rate is 8 *(each node will wake-up 8 times every second)*. Moreover, channel check rate values must always be a power of 2. For the simulations, protocols were tested with all the values *(which are powers of 2)* between 4 and 64. This range of channel check rate values, is sufficient for understanding how the network would behave in each case *(low, medium or high channel check rate)* and how channel check rate and consequently different duty cycles can affect a WSN in different network conditions. A more detailed representation of the configurations used during the experiments can be seen in Table 6.1.

Routing is handled by RPL [150] *(IPv6 Routing Protocol for Low power and Lossy Networks)*. Unit Disk Graph Medium (UDGM) in which nodes communicate and interfere in fixed-radius circles was used to emulate the radio environment in our simulations. The nodes used for the simulations were emulated Tmote sky motes. Emulated motes in Cooja simulate the same firmware as as would be uploaded and executed on a real Sky board.

Two different topologies were used for the simulations, a line topology of 9 nodes Figure 6.8 *(1 source, 1 sink, 7 intermediate nodes. Different simulation runs with different distance between sink and source)* and a 25 node random topology Figure 6.9 *(6 sources, 1 sink, 18 intermediate nodes)*. The first topology aimed at presenting each protocols performance in a low density network as well

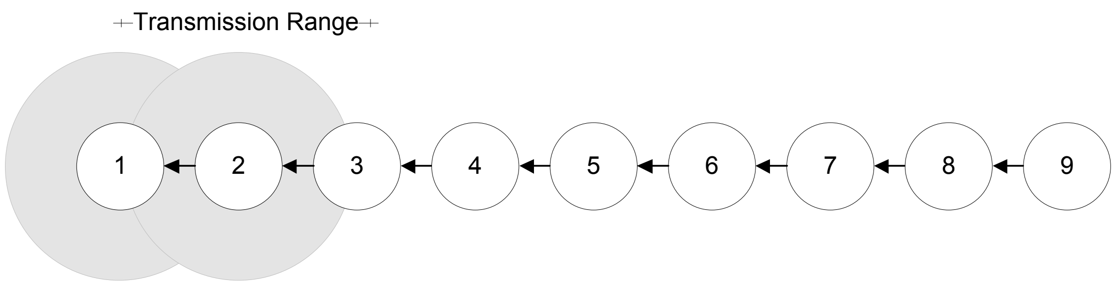

Figure 6.8: Routing tree with coverage *(line topology)*

as how the distance *(in hops)* affects the performance of each protocol. The aim of the simulations with random topology was to investigate the performance of the compared protocols in a WSN with higher node density and intense traffic load conditions.

Each experiment has been repeated 15 times with a new random seed per iteration. The duration of each experiment was 10 minutes. For each experiment we recorded the following metrics:

1. goodput as total number of packets received by the sink,
2. packet loss,
3. end to end delay (from source to sink),
4. energy consumption.

Lastly, we recorded the code footprint and memory requirements for each MAC/RDC configurations.

6.5.2 Simulation Results

6.5.2.1 Goodput

Figure 6.10 and Figure 6.14 are presenting a detailed protocol performance in terms of goodput for every simulation configuration *(different channel check rates and packet transmission intervals)*. The former represents the performance of the compared protocols during line topology simulations while the later represents the performance in the random topology. Additionaly, Figure 6.15 demonstrates the detailed goodput performance of the schemes for the different distances in hops during line topology simulations. Figure 6.12 describes how different channel check rate configurations affect goodput in a heavily congested WSN *(62.5 ms packet transmission interval)*. Finally, in Figure 6.13 a comparison of each protocols best performance in goodput over the different packet transmission intervals can be seen *(CADC&BEAM default performance since there is only one configuration for the channel check rate)*.

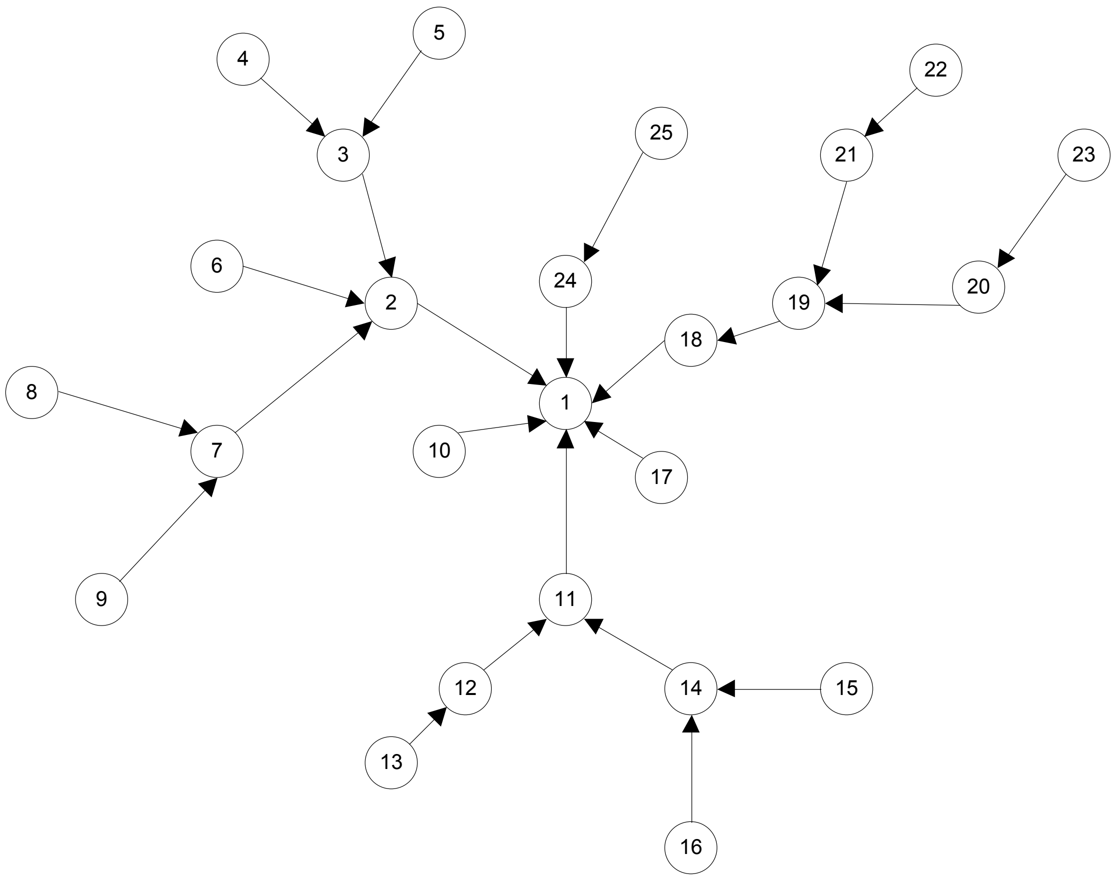

Figure 6.9: routing tree *(random 25 node topology)*

Further studying the above Figures, it can be observed that CADC and ContikiMAC outperformed X-MAC and BEAM protocols in terms of goodput. CADC's achieved a very high goodput, similar to the highest channel check rate configuration of ContikiMAC. In the line topology experiments CADC's goodput was lower than ContikiMAC's highest channel check rate configuration *(channel check 64 had the highest goodput for ContikiMAC)* by 6-10%. This result was expected since the initial channel check rate of CADC was 4. Therefore, CADC needs to adapt to the optimal channel check rate based on the traffic load in the network. This adaptation procedure can in turn take some time and thus lead to some packet loses while the sceme operates in lower than the optimal channel check rate. It is worthy to mention that the adaptation times of CADC will vary based on the traffic patterns and load *(bi-directinal traffic may take longer since nodes adjust the channel check rates in bi-direction flows. Moreover, based on the load CADC may or may not enter congestion collapse state which quadruples the channel check rate. In that case CADC will have lower adaptation times).*

On the contrary, during the random topology simulations, CADC's performance was higher than ContikiMAC's highest channel check rate configuration by 2-3%. Based on these results we can conclude that when network density and the traffic load increased, CADC outperformed ContikiMAC when configured with it's highest channel check rate. The main reasons behind this are:

- During random topology experiments, multiple sources may share the same traffic paths. Since the traffic was uni-directional, CADC mainly adapted the rates in nodes sharing the different flows. Moreover, in both experiments the sources were transmitting in a constant bit rate *(CBR)* and thus CADC didn't require frequent channel check rate adaptations. As mentioned during the previous paragraph, CADC may suffer some packet losses during it's channel check adapting phase. Since the sources had CBR traffic, CADC required approximately the same time to adapt in both scenarios and thus had the same packet loss. Having more sources in random topology simulations lead to a greater number of packets transmitted in the network. Since CADC lost approximately the same number of packets in both topology experiments but had significantly more packets transmitted during the random topology; the loss due to CADC's adaptation periods was much less noticeable.

- the modified back off algorithm in CADC, which is more intelligent and adds more randomness to the back off algorithm.

- In CADC, each node may operate in a different channel check rate from its neighbour nodes. Each node calculates the maximum number of packets that it can transmit every second based on its Channel check rate and thus it is directly connected with the maximum size of the contention window *(CW)* calculated by the back off retransmission algorith *(similarly to CSMA)*. Consequently, when neighbour nodes have different size CW the probability of collisions is further reduced since the back off interval can be between 0 and CW.

- Random topology simulations had a greater number of source nodes and thus some of these sources shared the same traffic path to the source. When a traffic path is shared by multiple sources, it is more likely that CADC will enter the heavy congestion state and significantly increase the channel check rates at the nodes. This will result in less channel check rate adaptations in the same path and consequently faster adaptation to the high traffic demands. The two above mentioned characteristics of CADC is described in detail in section 6.3.

It can also be observed that CADC and ContikiMAC managed to maintain high goodput for all the simulated packet transmission rates. On the contrary, packet transmission rate had a significant impact to the performance of X-MAC and BEAM. From the simulations it is noticeable that X-MAC and BEAM, demonstrated similar to CADC's and ContikiMAC's goodput performance when the traffic in the network was low. On the other hand, during high traffic configurations their performance was significantly lower. If we further analyze the performance of X-MAC, when the channel check rate was set to 4, a tenfold decrease *(between packet transmission intervals of 62.5 and 500ms)* in the protocols performance can be observed. Based on the above, it is easy to conclude that the packet transmission mechanism with probing used by BEAM and X-MAC is much more susceptible to congestion.

Furthermore, it is noticeable that when the distance *(in hops)* between the source and the sink increased, CADC achieved the most stable behavior. CADC's goodput dropped by 24% when we increased the distance by 8 hops. On the other hand ContikiMAC's goodput droped by 30%, X-MAC's by 55% and BEAM's by 43%. How the distance in hops affected the goodput of each mechanism can be seen in Figure 6.11.

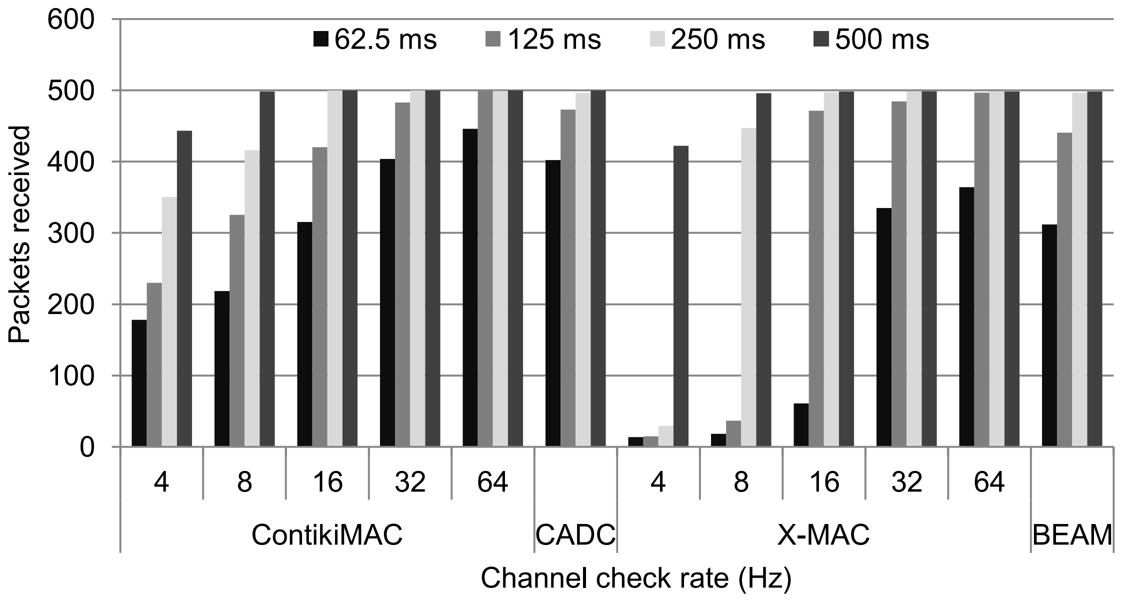

Figure 6.10: Goodput for the different packet transmission intervals with different channel check rates *(line topology)*

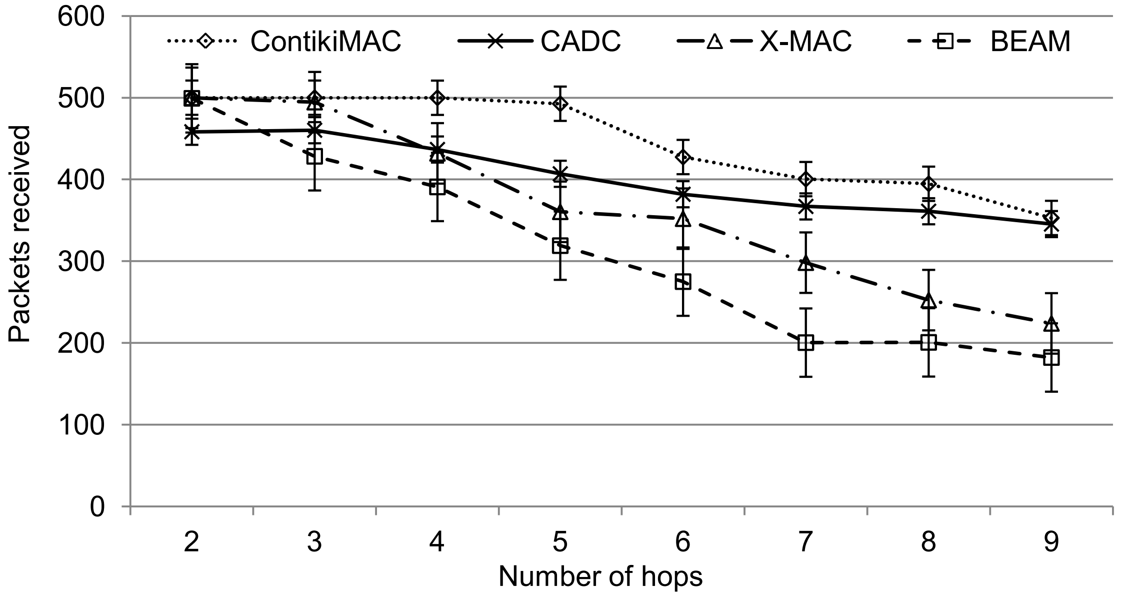

Figure 6.11: Goodput over distance in hops (ContikiMAC&X-MAC best performance. *Channel check rate=64 - line topology*)

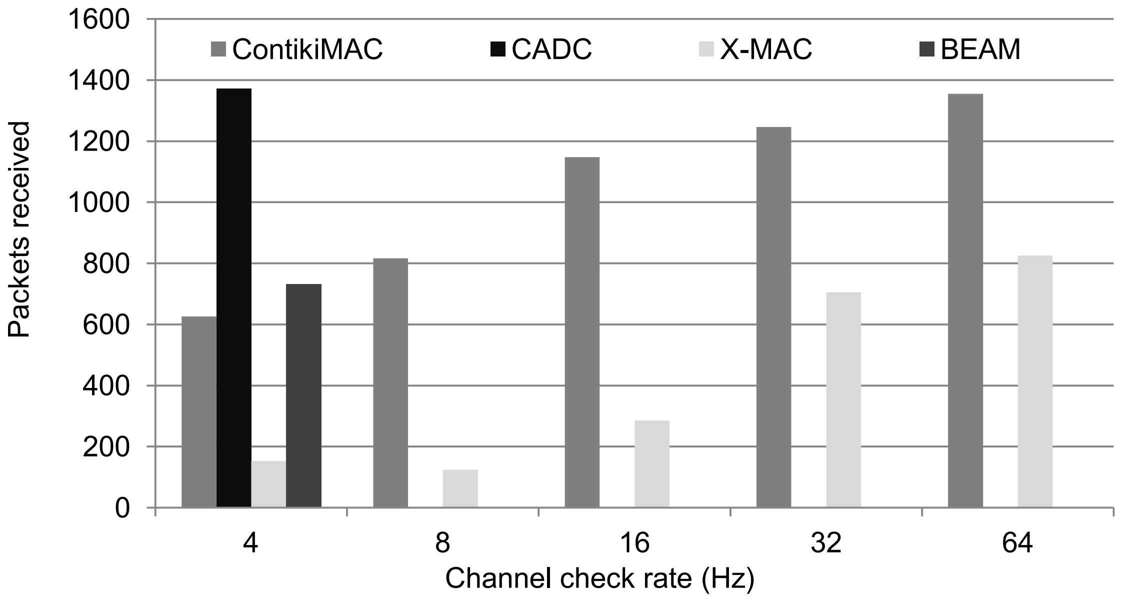

Figure 6.12: Goodput over the different channel check rates (*highest traffic condition with inter packet transmission interval equal to 62.5 ms - random topology)*

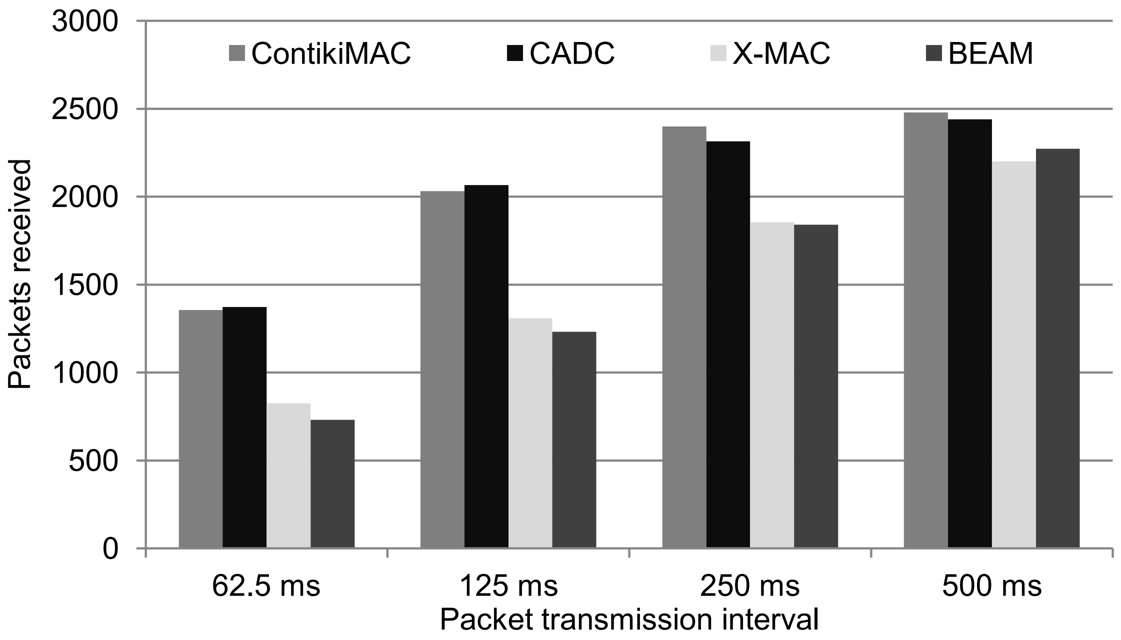

Figure 6.13: Goodput with different packet transmission intervals (ContikiMAC&X-MAC best performance. *Channel check rate=64 - random topology*)

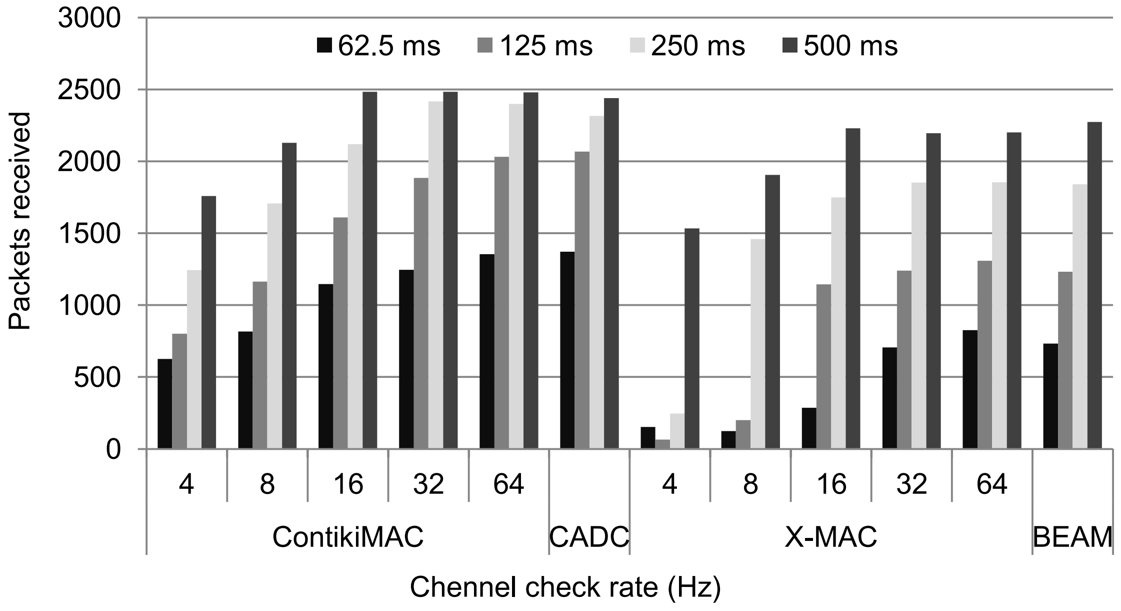

Figure 6.14: Goodput for the different packet transmission intervals with different channel check rates (*random topology*)

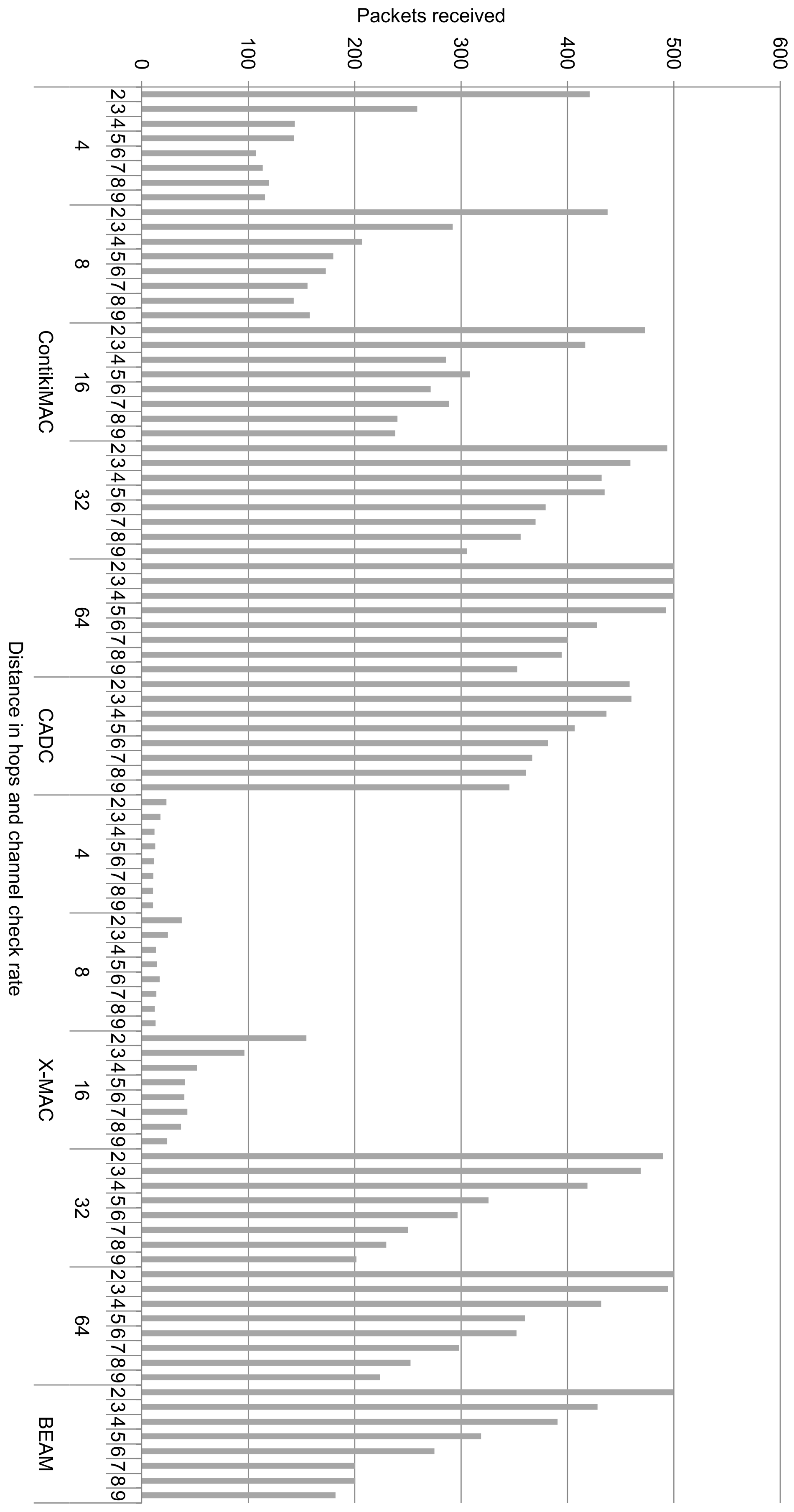

Figure 6.15: Goodput for the different channel check rates and distances in hops *(the demonstrated performance is during heavy congestion with inter packet transmission interval of 62.5 ms)*

6.5.2.2 Packet Loss

Figure 6.16 shows the average packet loss for the line topology simulations. Figure 6.17 presents a detailed analysis of the packet loss for the different transmission rates during random topology simulations. Similarly to goodput *(as application layer packet transmissions are not reliable)*, CADC and ContikiMAC achieved lower loss for both simulation scenarios. In comparison to line topology, both CADC and BEAM had better performance compared to ContikiMAC and X-MAC when the network density and traffic load increased.

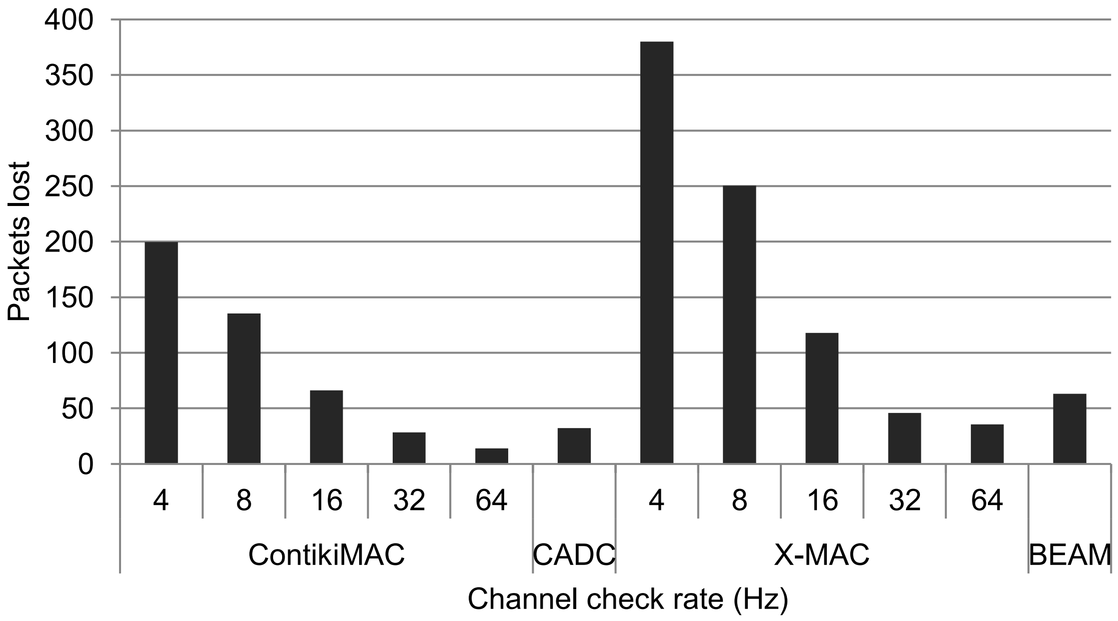

Figure 6.16: Average packet loss (*line topology*)

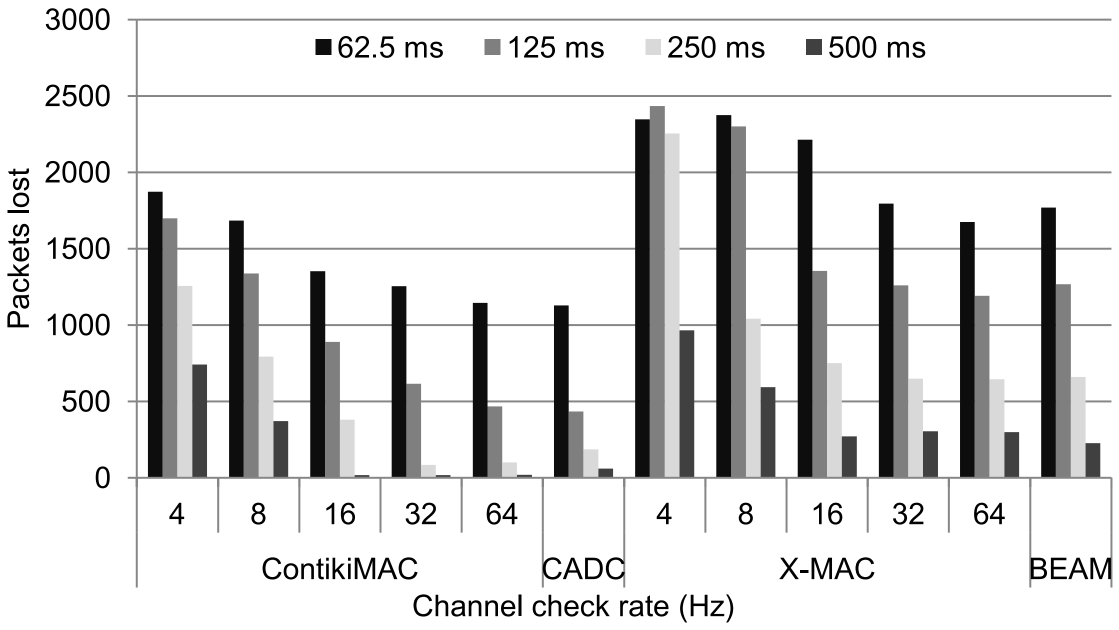

Figure 6.17: Packet loss for the different packet transmission intervals with different channel check rates (*random topology*)

6.5.2.3 Packet Delay

Figure 6.19 shows packet delay against the distance in hops. Overall, X-MAC configured with a high channel check rate had the lowest delay in the majority of the simulated experiments with BEAM following and lastly ContikiMAC and CADC. When the network was configured with BEAM or X-MAC, the goodput was lower and thus there was less packets in the network. Having fewer packets in the network results in less built up queues and thus lower delay times. Even though X-MAC had the lowest delay times, both BEAM and CADC maintained lower delay than X-MAC and ContikiMAC when the later was configured with lower channel check rates and thus duty cycles. It can also be observed that CADC achieved lower delay than any channel check configurations of ContikiMAC and X-MAC between 4 - 16. BEAM demonstrated an even better delay with lower delay times for any channel check configurations of ContikiMAC and X-MAC between 4 - 32. The average delay for the line topology simulations can be seen in Figure 6.18 while Figure 6.20 shows the detailed delay of each mechanism for the different simulation configurations during random topology experiments. Additionally to the above, Figure 6.19 shows how delay scales when the distance in hops increases while Figure 6.22 and Figure 6.23 demonstrate the delay times for the different channel check rates over distance in hops for the line topology simulations.

Further analysis of the delay in the simulation experiments revealed that ContikiMAC's and X-MAC's delay was reduced as the packet transmission rate decreased *(inter packet transmission interval increases)*. This is expected since higher transmission rates contribute to built up of the packet queue. On the contrary, CADC and BEAM in some cases demonstrated higher delay when the packet transmission rate was lower. Figure 6.21 is a good example to study the above observation. In this figure it can be observed that the average delay times for CADC and BEAM varied for the different inter packet transmission intervals. More specifically, CADC shown much higher delay times when the sources were transmitting one packet every 125ms. As explained in section 6.3, a dynamic duty cycle protocol adjusts its cycle based on the traffic parameters. Therefore, in the same topology when the packet transmission rate is higher CADC and BEAM may adjust the nodes channel check rate to a higher value *(higher frequency)*. Higher channel check rates result in a greater number of layer 2 packet transmission rate and thus a faster drain of the packet queue, which in turn results in a lower delay times. Taking the above under consideration the jitter observed during the delay times for the various source rates, when the network was configured with CADC and BEAM, can be explained.

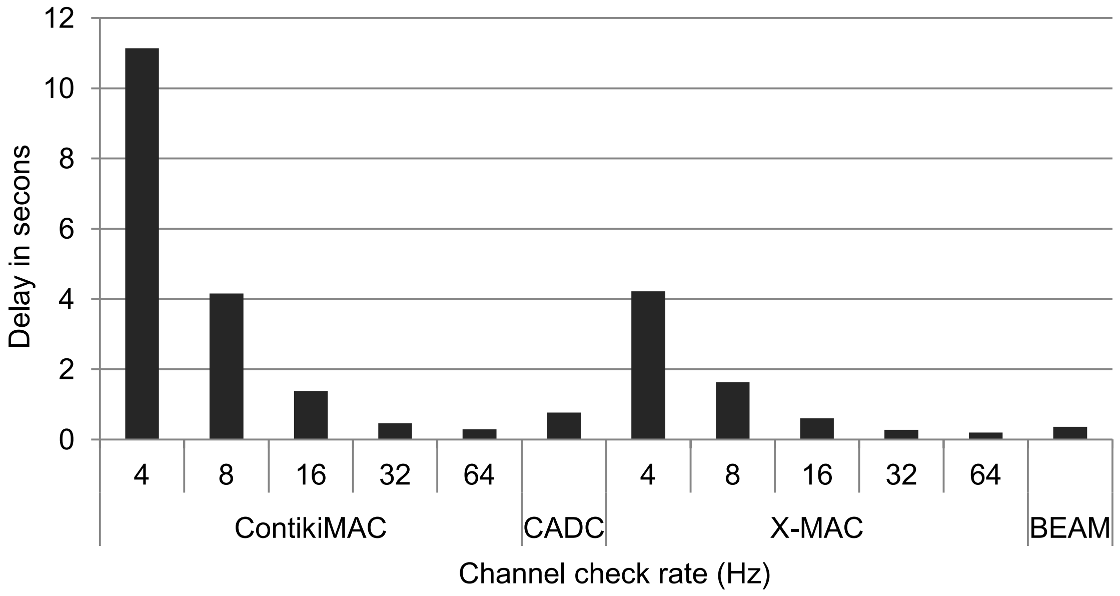

Figure 6.18: Average delay with different channel check rates *(line topology)*

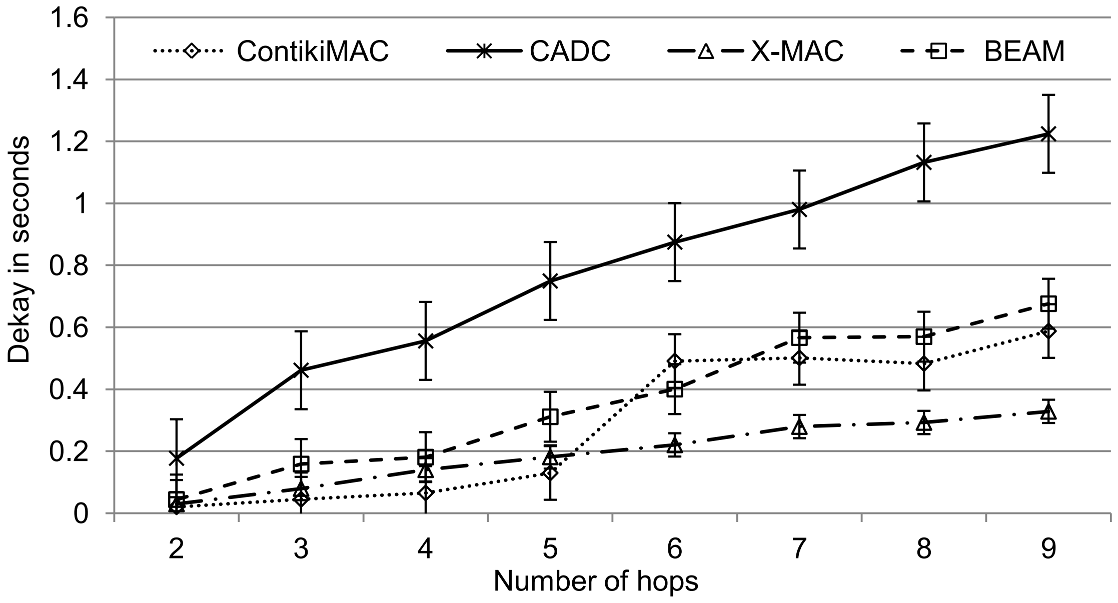

Figure 6.19: Packet delay over distance in hops (contikiMAC&X-MAC best performance. *Channel check rate=64 - line topology*)

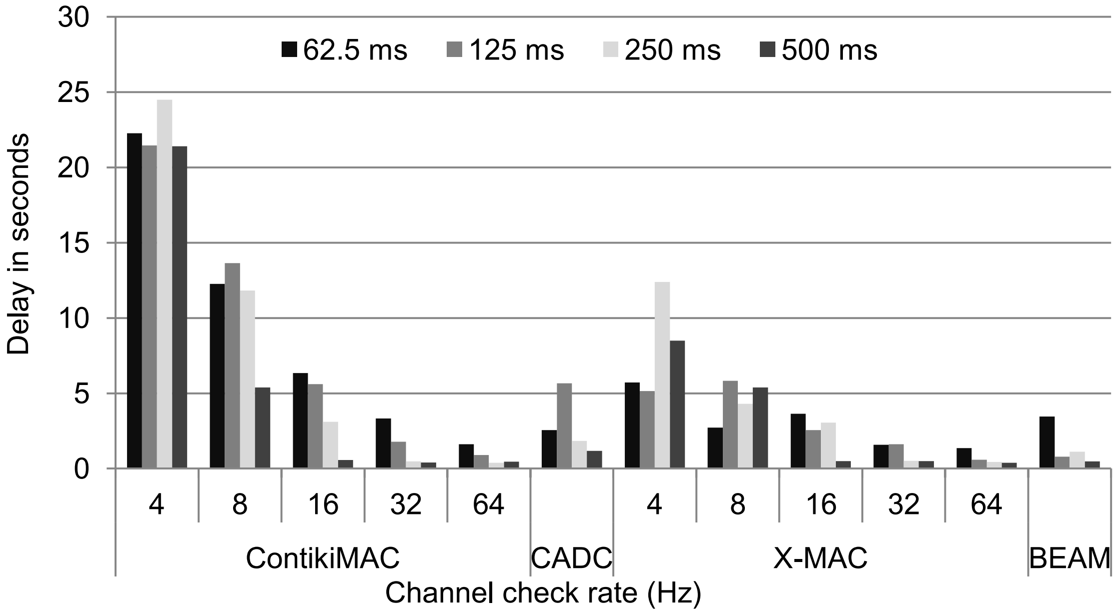

Figure 6.20: Average packet delay for the different packet transmission intervals with different channel check rates *(random topology)*

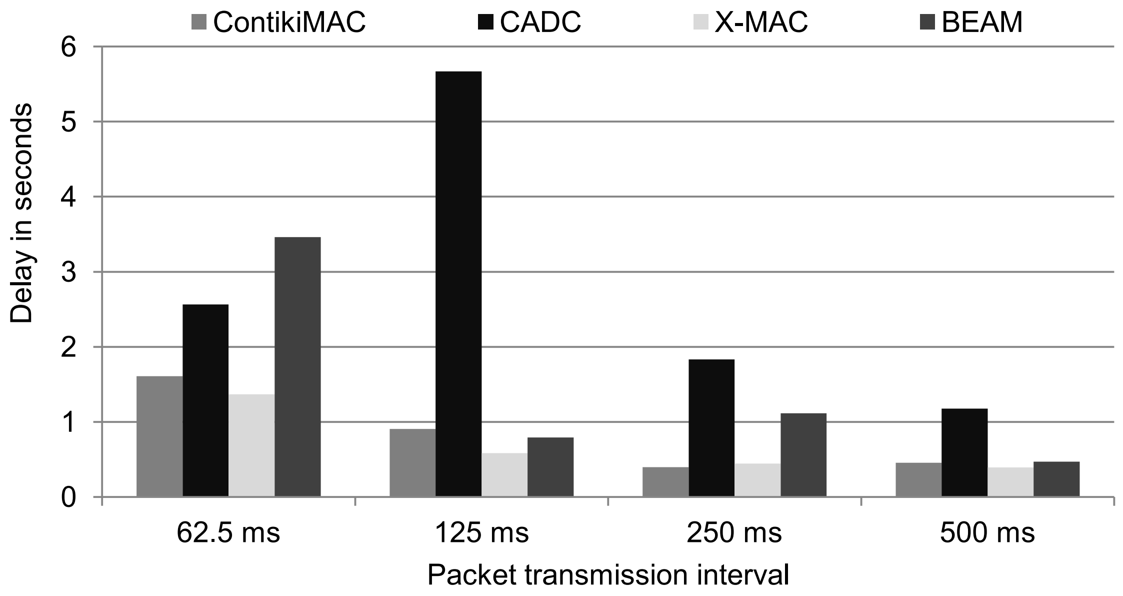

Figure 6.21: Packet delay with different packet transmission intervals (contikiMAC&X-MAC best performance. *Channel check rate=64 - random topology*)

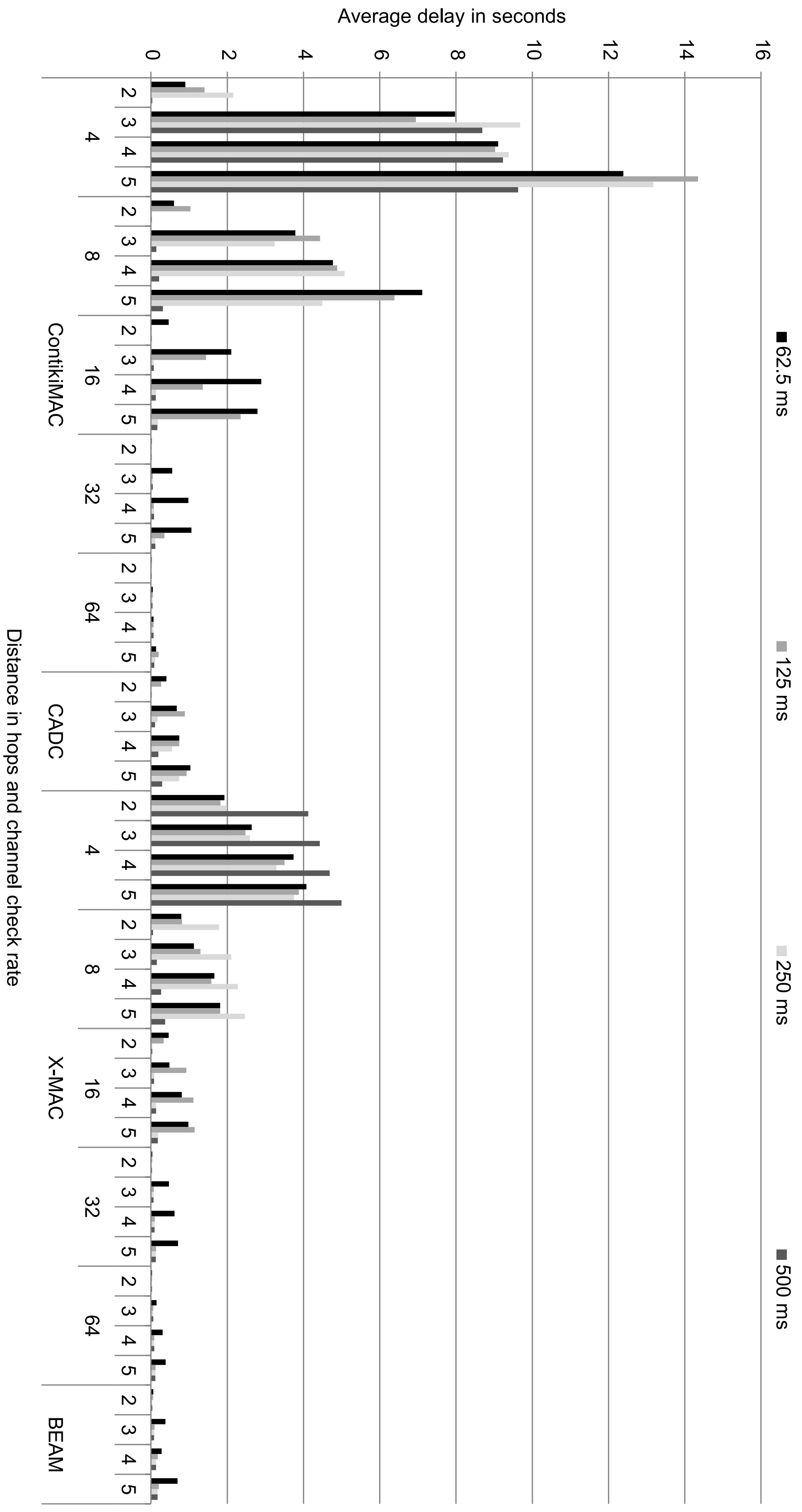

Figure 6.22: Packet delay for the different channel check rates and distances between 2 - 5 hops *(line topology)*

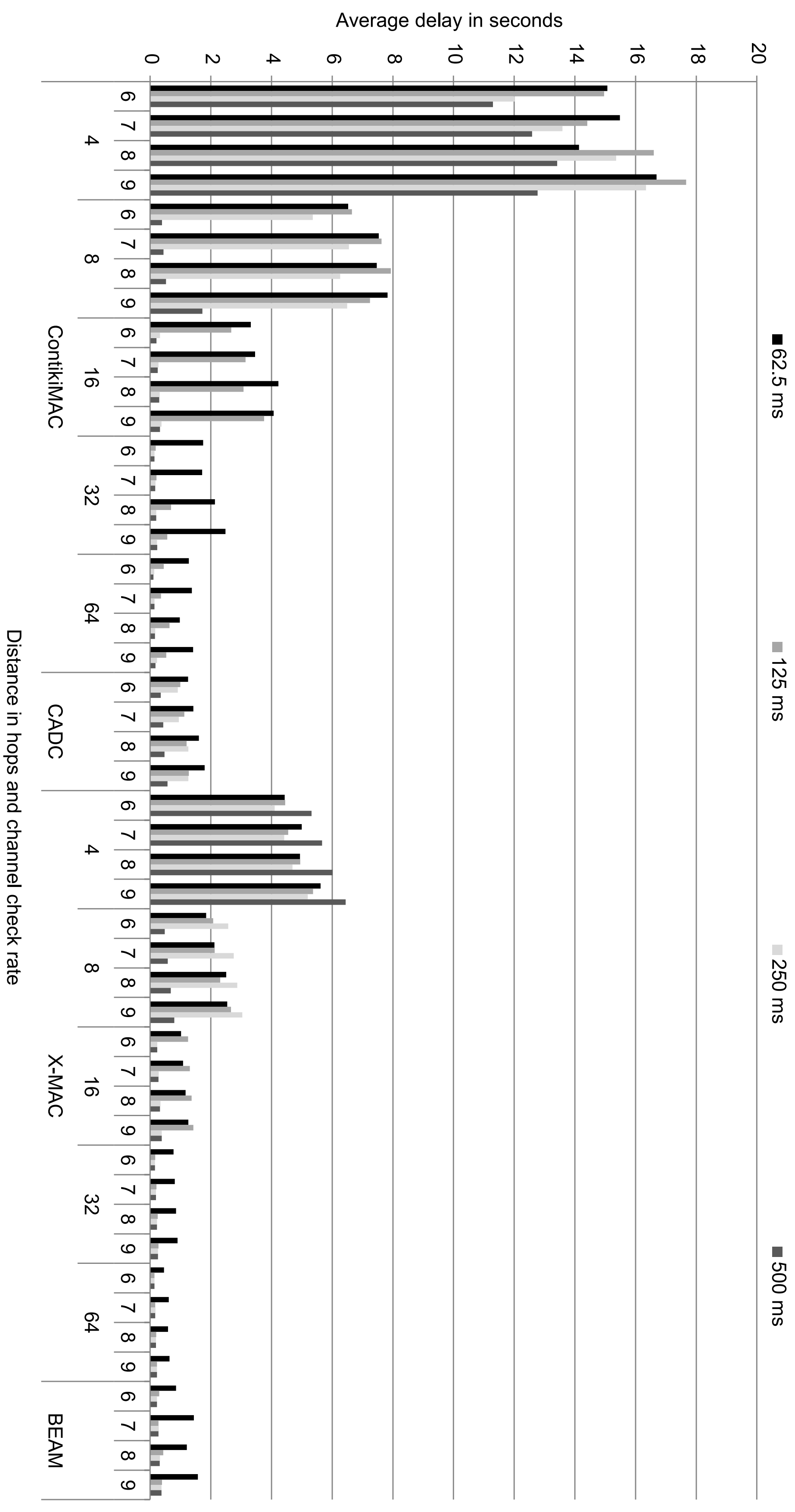

Figure 6.23: Packet delay for the different channel check rates and distances between 6 - 9 hops *(line topology)*

6.5.2.4 Energy Consumption

Figure 6.24, demonstrates the per node average energy consumption per second when the network is idle *(no packet transmissions)*. It can be observed, that when the network is idle, the per node energy consumption approximately doubles when the channel check rate is doubled *(as mentioned in the beginning of this section channel check values can only be powers of 2)*. In ContikiMAC and X-MAC the channel check rate must be pre-configured. The default channel check rate value for the majority of the platforms in Contiki OS is 8. In order to increase the bandwidth, a WSN can be configured with a higher channel check rate. Configuring a WSN with a channel check rate of 64 will result in an approximately 800% higher energy consumptin compared to the default channel check value of 8.

Figure 6.25 shows the average energy consumption per received packet, for the line topology simulations. In some cases, it is also noticeable that when the network is active, a higher channel check rate configuration can result in lower energy consumption. Studying Figure 6.28 and Figure 6.29, this phenomenon can be explained by comparing the average energy consumption in the network with the average energy consumption per successfully received packet. The former Figure, represents the average node energy consumption for each node consumed during random topology simulations while the later represents the per packet energy consumption for each node.

The total average energy consumption for X-MAC protocol was followed using the same pattern as the energy consumption when the network was idle. Higher channel check rates resulted in significantly higher energy consumption. On the contrary, the average energy consumption per successfully received packet didn't follow the same pattern for every transmission rate. The cause of this was the high packet loss of X-MAC in high packet transmission rates. On the other hand, ContikiMAC exhibited higher over all energy consumption for the two lowest values of the channel check rate *(4–8)*. This can more distinctively be observed during low packet transmission rates *(longer packet transmission intervals. See Figure 6.28)* since higher rates had significantly higher packet loss *(higher packet loss results in less total transmissions from the nodes in the network)*. When the network is idle, ContikiMAC's energy consumption is very low compared to the energy consumed during packet transmissions or packet reception/idle listen. Moreover, the duration of each packet transmission in ContikiMAC is longer for lower channel check rates *(a packet is continuously transmitted for a period of time or until an acknowledgment is received. This period of time is calculated based on how often each node scans the radio for packet transmissions)*. The above characteristics of ContikiMAC result in higher energy consumption per packet transmission for lower channel check rates. Therefore, when the network transmits the same number of packets with a lower channel check rate, the energy consumption is going to be greater than a higher channel check rate configuration. Even though the same principle applies to X-MAC, the above phenomenon is not observed with that scheme due to X-MAC's significantly higher energy consumption when the network is idle. Therefore, the amount of extra energy consumed per transmission is too small to be observed compared to X-MAC's overall energy consumption.

CADC and BEAM maintained the lowest possible energy consumption when the network was idle with overall *(for every distance in hops and packet transmission interval)*, BEAM achieving approximately 40% lower energy consumption than X-MAC's configuration with the lowest energy consumption. Furthermore, CADC achieved the lowest energy consumption in every simulated scenario. CADC's overall energy consummption per succesfully received packet was lower than ContikiMAC's lowest energy consumption configuration by 20% during line topology experiments and approximately 15% lower during random topology experiments. Figure 6.26 shows how energy consumption increases over the distance while Figure 6.31 demonstrates the energy consumption per succesfully received packet for the different distances in hops. In Figure 6.27 average energy consumption over different packet transmission intervals is plotted.

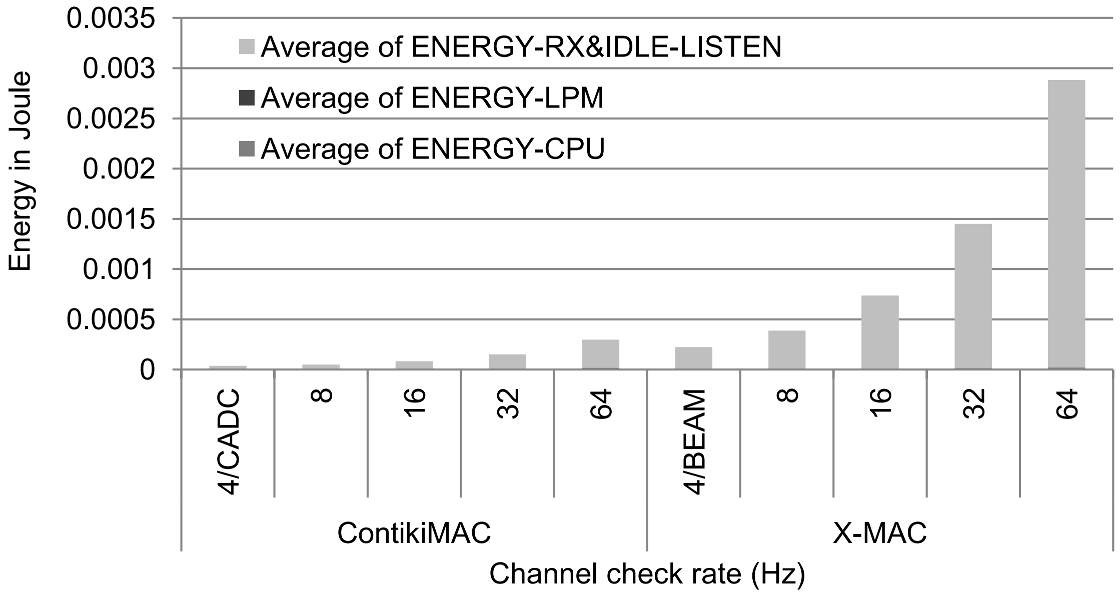

Figure 6.24: Idle network energy consumption/sec for each node

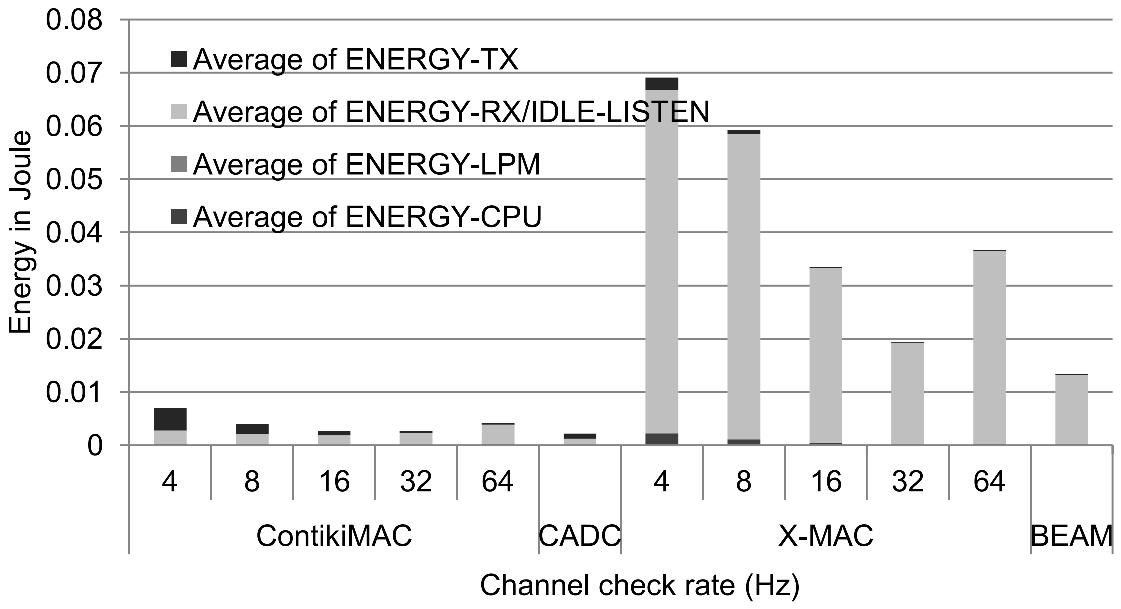

Figure 6.25: Average energy consumption per successfully received packet *(line topology)*

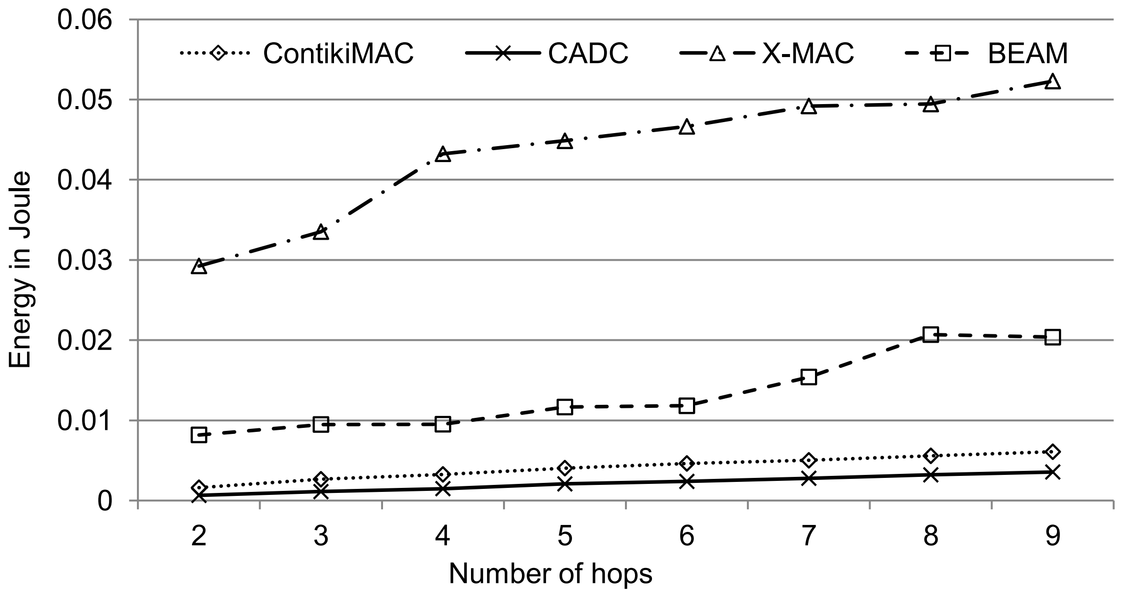

Figure 6.26: Average energy consumption/successfully received packet over distance in hops *(for all channel check rates, line topology)*

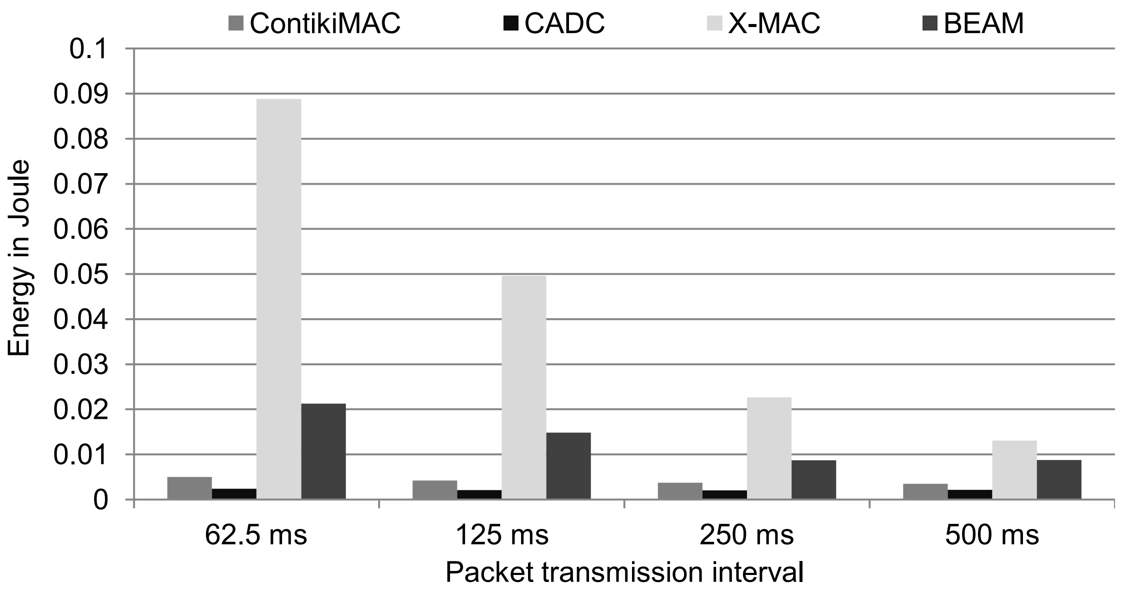

Figure 6.27: Average energy consumption/successfully received packet with different packet transmission intervals *(line topology)*

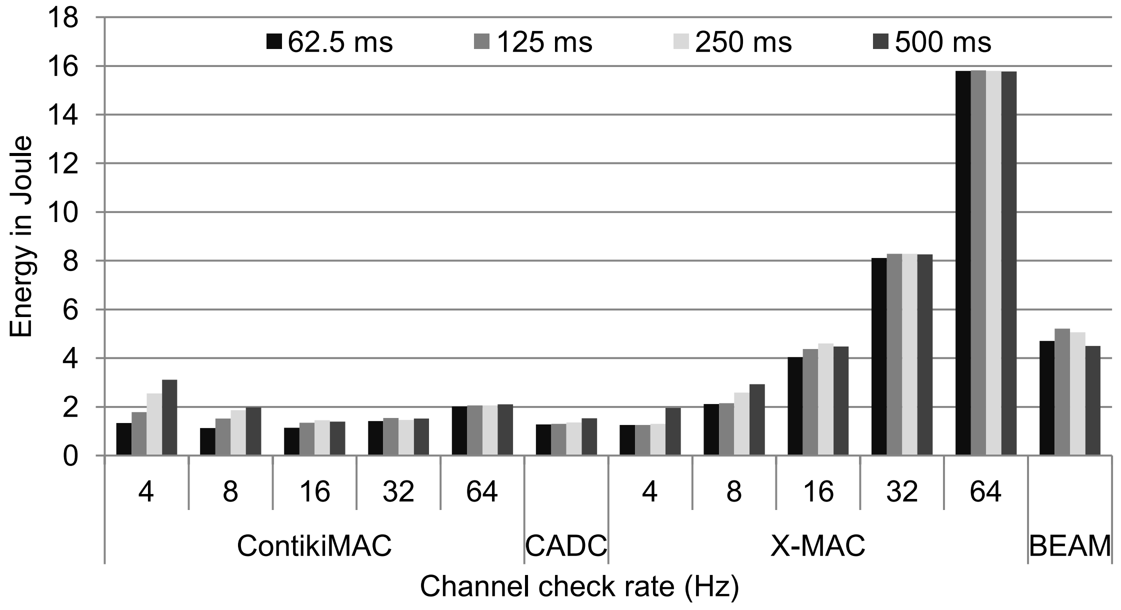

Figure 6.28: Total energy consumption for the different packet transmission intervals with different channel check rates *(random topology)*

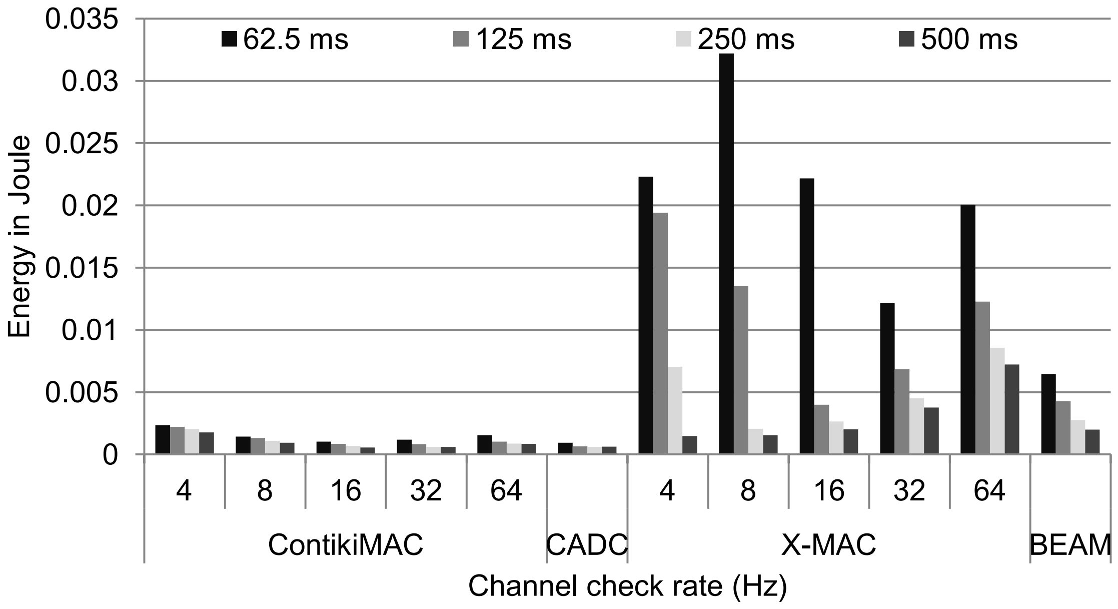

Figure 6.29: Energy consumption/successfully received packet for the different packet transmission intervals with different channel check rates *(random topology)*

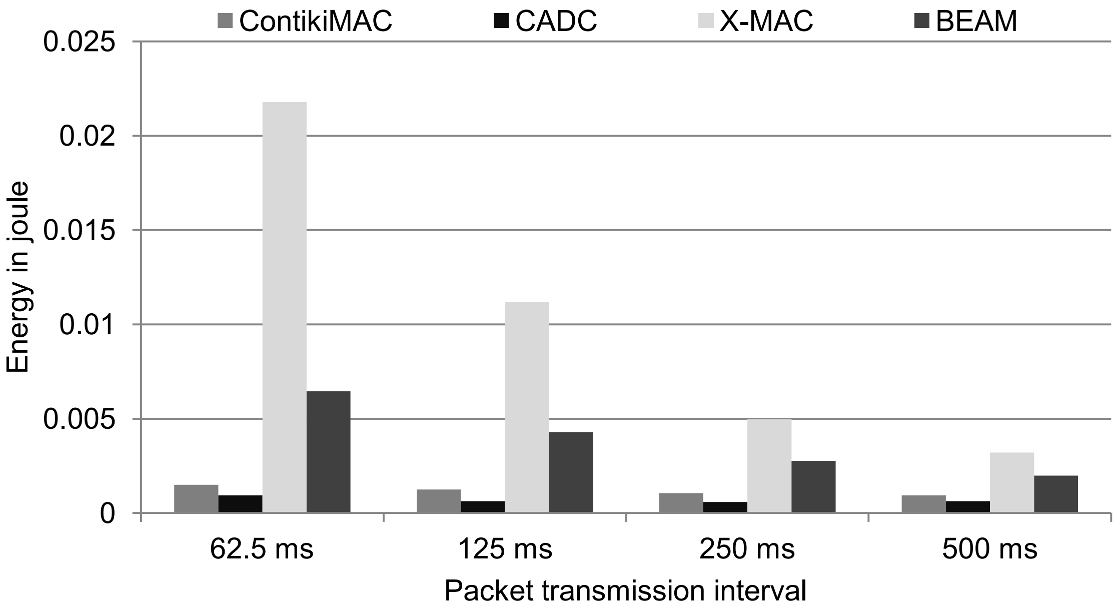

Figure 6.30: Average energy consumption per successfully received packet with different packet transmission intervals *(random topology)*

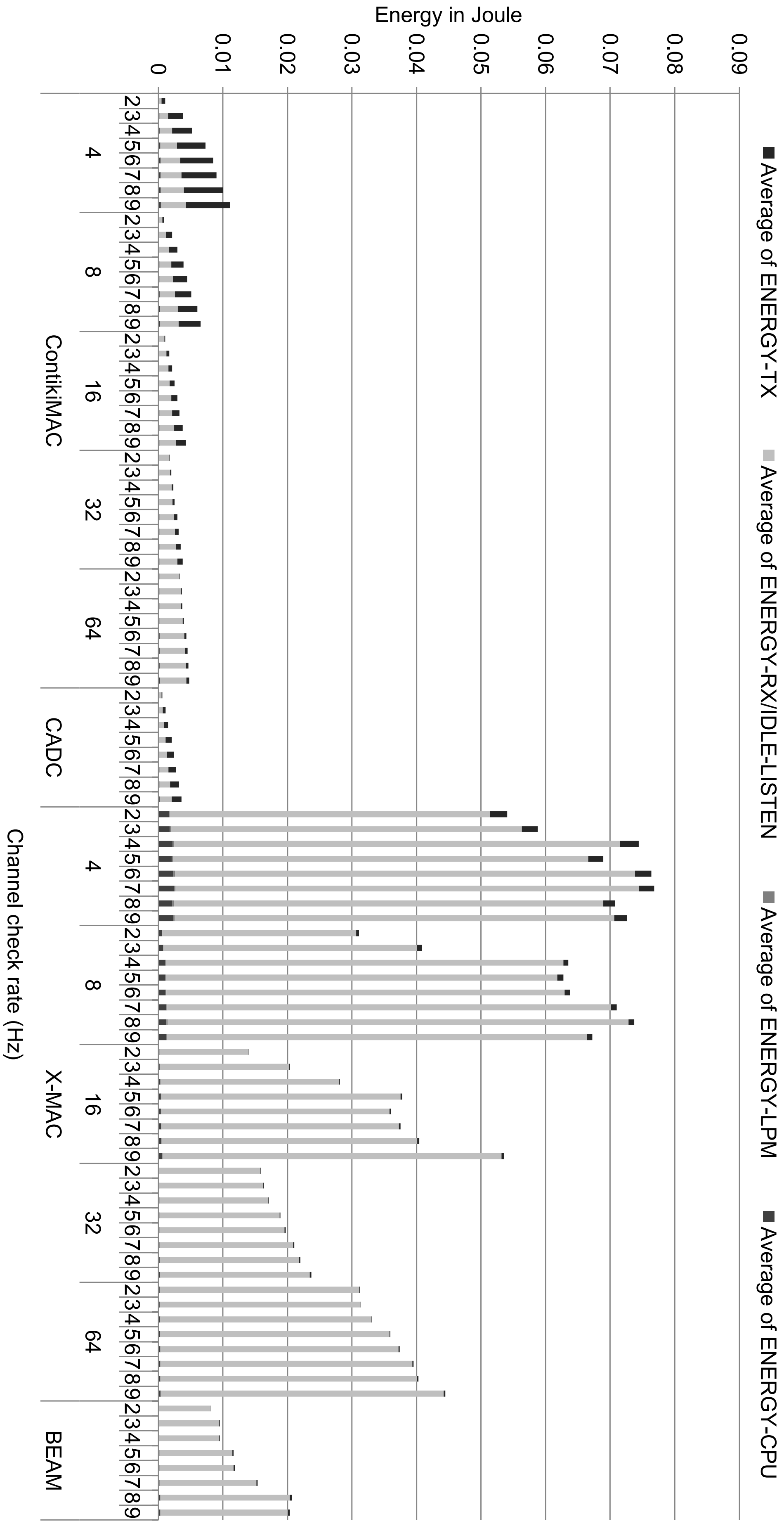

Figure 6.31: Average energy consumption/successfully received packet for the different distances in hops *(line topology)*

6.5.2.5 Memory Requirements

Table 6.2 show the memory requirements of the compared protocols. The numbers in this table represent the size of each algorithm in both MAC and RDC layer. X-MAC has the lowest memory requirements while CADC requires approximately 1kb extra memory than the rest of the protocols. This was expected since CADC incorporates much more functionalities in order to precisely calculate the desired cycle at each given time. Actual memory requirement will depend on the architecture of the implementation platform, but the relative values here will be relevant in most platforms. During section 6.6, the sizes of the protocols compiled with SDCC are going to be presented while a comparison thereof will be made in section 6.7.

Table 6.2: Size of protocols in bytes *(RDC + MAC layer, compiled with MSP430 GCC)*

	CADC	ContikiMAC/ CSMA	X-MAC/ CSMA	BEAM
text	3236	2378	2030	2544
data	28	24	38	38
bss	420	366	250	274
dec	3684	2768	2318	2856

6.5.2.6 Simulation Result Analysis

The main objective in this research study is to show the impact of duty cycles on the performance of WSNs. As shown in previous sections, increased duty cycles *(higher channel check rates)* significantly increase the performance of WSN in terms of goodput, delay and packet. Furthermore it is observed that increased duty cycles *(in %)* does not necessarily lead to higher energy consumption when the network is active *(see ContikiMAC energy performance in subsubsection 6.5.2.4)*. On the contrary, increased duty cycles have significantly higher energy consumption when the sensor network is idle. As explained in subsubsection 6.5.2.4 when the network is idle, energy consumption is approximately doubled for each increase *(powers of 2)* of the channel check rate. Assuming that a WSN is configured in a lower channel check rate, its lifespan can be extended for months. Moreover, experiments have shown that for the different sensor network conditions *(load of traffic and distance)*, different duty cycle configurations had the optimal performance in terms of energy consumption, goodput, delay and packet loss. To be more precise, ContikiMAC's energy consumption per succesfully received packets varied between the different packet transmission intervals and channel check configurations. Based on the above, in order to achieve the best performance with the lowest energy consumption in a WSN, nodes should be frequently reconfigured. Frequent reprogramming in large scale sensor network implementations is difficult and energy consuming *(in case over the air reprogramming)*. The decision of the optimal duty cycle for a sensor network can be even harder in 6LoWPAN where the communication patterns are arbitrary and multiple different applications may be operating at same WSN *(different packet sizes, bandwidth requirements and destination nodes)*. This problem can be solved with the use of protocols such as CADC and BEAM which dynamically adjust the duty cycle in the nodes in order to minimise energy consumption and maximise the performance. Simulation results have shown that the proposed protocol, successfully coped with the different network parameters during our simulations and adjusted the duty cycles in each node accordingly. Furthermore, CADC achieved close to the highest possible goodput and the lowest possible packet losses, the lowest energy consumption and very competitive delay times.

6.6 Test-bed Experiments

6.6.1 Test-bed Configurations

In this section, the performance of the default DC protocol in Contiki is investigated and compared with CADC through test-bed experiments. All of CADC features described in section 6.3 having been implemented as a new MAC/RDC layer for the Contiki embedded operating system. Similarly with the simulation experiments, test-bed aimed at evaluating the behaviour of the protocols in both heavily congested and normal network conditions. In order to achieve this, we conducted approximately 720 testbed experiments, including multiple runs with different permutations of channel check rate, inter packet transmission interval and distance in hops. The different packet transmission intervals *(the time between two consequent packet transmissions)* used for the comparison of the mechanisms during our experiments were: 250ms, 125ms and 62.5ms, with each packet having a 24-byte application data. In each experiment, the source attempted to transmit 100 packets. The configurations of packet queue and routing protocol used during the test-bed experiments were identical to the simulation runs.

In Contiki OS and hence our experiments, the radio duty cycle is expressed as a function of the wake-up frequency. In our results and graphs this frequency is called channel check rate. As mentioned earlier in section 6.5 for the majority

Table 6.3: Different testbed configuration permutations

	ContikiMAC	CADC
MAC layer	CSMA	CADC
RDC layer	ContikiMAC	CADC
Channel check rate	8 - 16 - 32 - 64	8 *(dynamic adjustment)*
min_{CCR} *(Hz)*	configuration dependent	8
max_{CCR} *(Hz)*	configuration dependent	64
n_i	N/A	10
Q_h	N/A	90%
Q_l	N/A	60%
U_T	N/A	20%

of platforms supported in Contiki, the default CCR value is 8 *(each node will wake-up 8 times every second).* Moreover, channel check rate values must always be a power of 2. The range of channel check rate values used during our testbed experiments as well as the packet transmission interval and other configuration specific information can be seen in Table 6.3. The different testbed configuration permutations used was selected carefully and it is crucial for understanding how the network would behave in each case *(low, medium or high channel check rate)* and how channel check rate and consequently different duty cycles can affect a WSN in different network conditions.

Subsequently, we investigated the performance of CADC and ContikiMAC in a 15 node multihop indoor testbed *(Appendix D explains in detail why we picked 15 nodes for our test-bed).* Our deployment is based on Sensinode N740 NanoSensors (TI/Chipcon cc243x System-on-Chip with IEEE 802.15.4 low power RF transceiver, 8KB volatile RAM and 128KB Flash). For this work we used our port of the Contiki OS for Sensinode/cc2430 devices [145]. Similarly to the method described in section 3.11 the firmware on each device implements a UDP server, which waits listening for a packet which signals the start of a test. Upon reception, the device will extract experiment configuration parameters *(number of packets to be transmitted and inter-packet interval)* and start the test. By using this method, any of the devices can act as a traffic source with varied transmission lengths and source rates and thus we can test multiple different scenarios in the same deployment. Moreover, each sensor was periodically transmitting its energy consumption measurements. A USB border router, is used to route IPv6 traffic between the 6LoWPAN and the Internet. The receiver of the flows is an Internet host, controlling tests and performing all logging and necessary measurements. After switching the deployment on, the RPL routing protocol converged forming

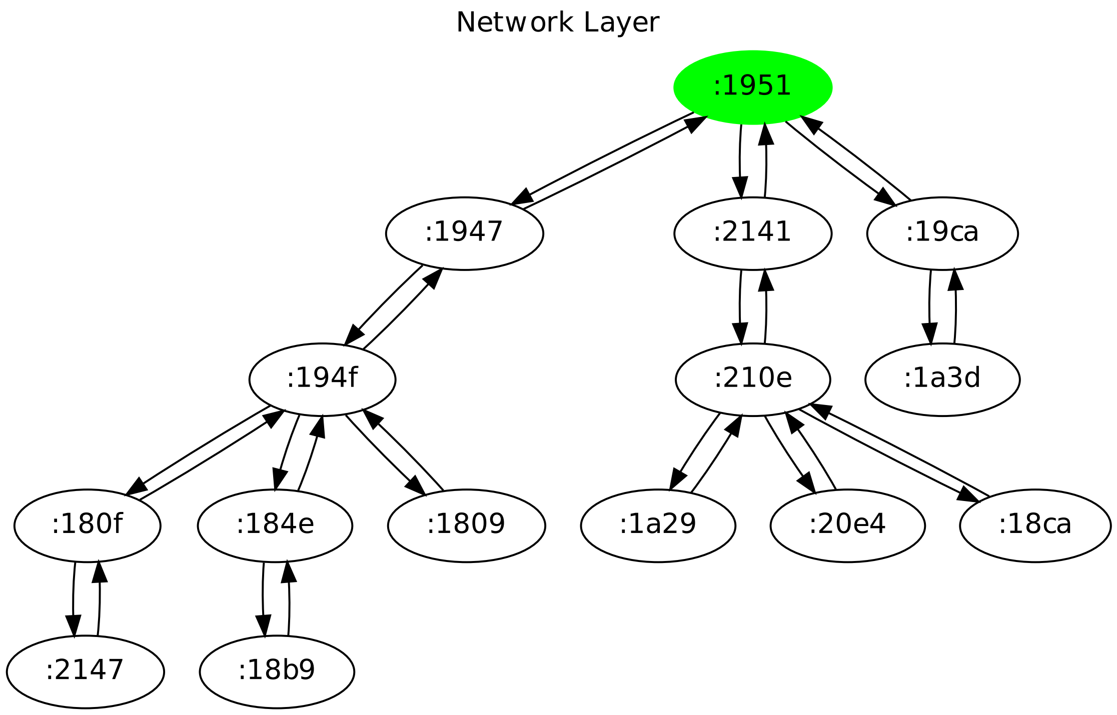

Figure 6.32: Routing tree created by RPL during our test bed experiments

trees similar to the one illustrated in Figure 6.32. The number inside each bubble depicts the last two bytes of each node's IPv6 address.

Each experiment was repeated 10 times. For each experiment we recorded the following metrics:

1. goodput as percentage of packets received by the sink over the total transmitted packets,
2. packet loss,
3. Round trip delay,
4. energy consumption.

Additionally, the code footprint and memory requirements for each MAC/RDC configuration is recorded.

Finally, it is worthy to mention that the test-bed scenarios was designed as close as possible to the simulation ones in order to achieve a result comparison between thereof.

6.6.2 Testbed Results

6.6.2.1 Goodput

Figure 6.33 is presenting in detail the percentage ratio of the successfully received packets at the base station for every different permutation of channel check

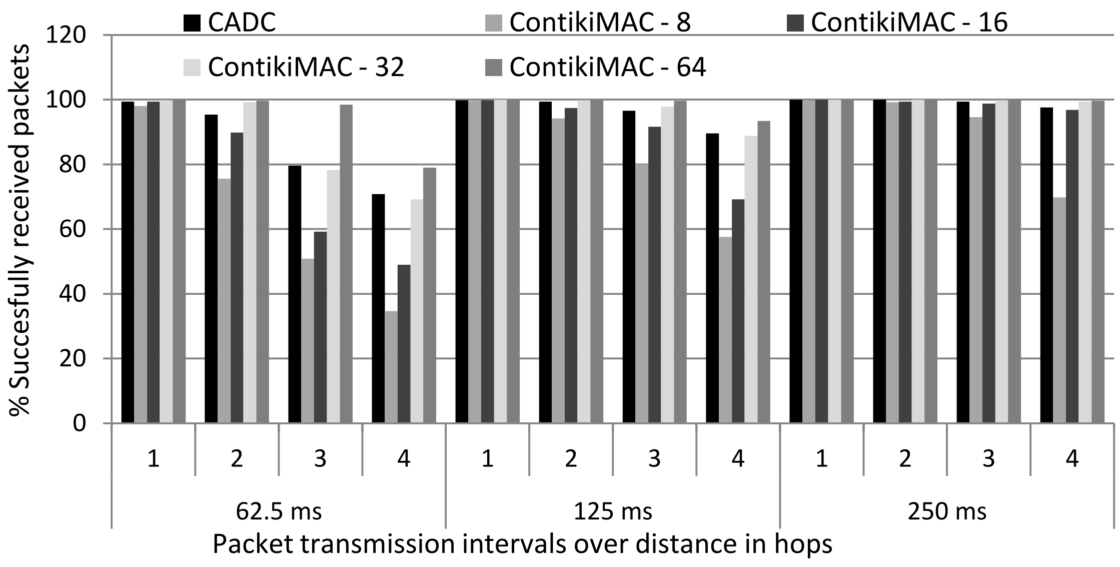

Figure 6.33: Goodput for different inter packet transmission intervals over channel check rate and distance in hops

rate and packet transmission interval. Further studying this figure, it can be observed that both distance and different packet transmission intervals affect the performance of the protocols. It can also be observed, that the higher the channel check rate of ContikiMAC the best goodput the protocol achieved. Between the different protocol configurations, ContikiMAC with channel check rate of 8 demonstrated a significantly inferior goodput in every possible combination of distance and packet transmission interval. The reason behind this poor performance was that all the combinations of packet transmissions intervals and network density *(most of the nodes were in range of each other)* used during the experiments was enough to cause intense congestion when ContikiMAC was configured with a channel check rate of 8. Both CADC and ContikiMAC with channel check configurations between 16 - 64 didn't face any congestion problems and achieved optimal performance during light traffic *(250ms packet interval)*. On the contrary, the more we decreased the packet transmission interval the more visible became the difference between the performances of the different channel check rate configurations. This phenomenon is magnified when the packets were injected to the network in 62.5 ms intervals and thus this configuration would be ideal for analysing the performance of the protocols during congestion.

Figure 6.34 shows in detail the performance of the protocols in terms of percentage goodput over distance in hops during high traffic *(62.5 ms packet interval)*. From this figure, it can be observed that CADC had lower goodput than ContikiMAC when configured with a channel check rate of 64 but achieved higher goodput than the remaining ContikiMAC configurations. As mentioned in section 6.3, CADC is dynamically adjusting its channel check rate based on the

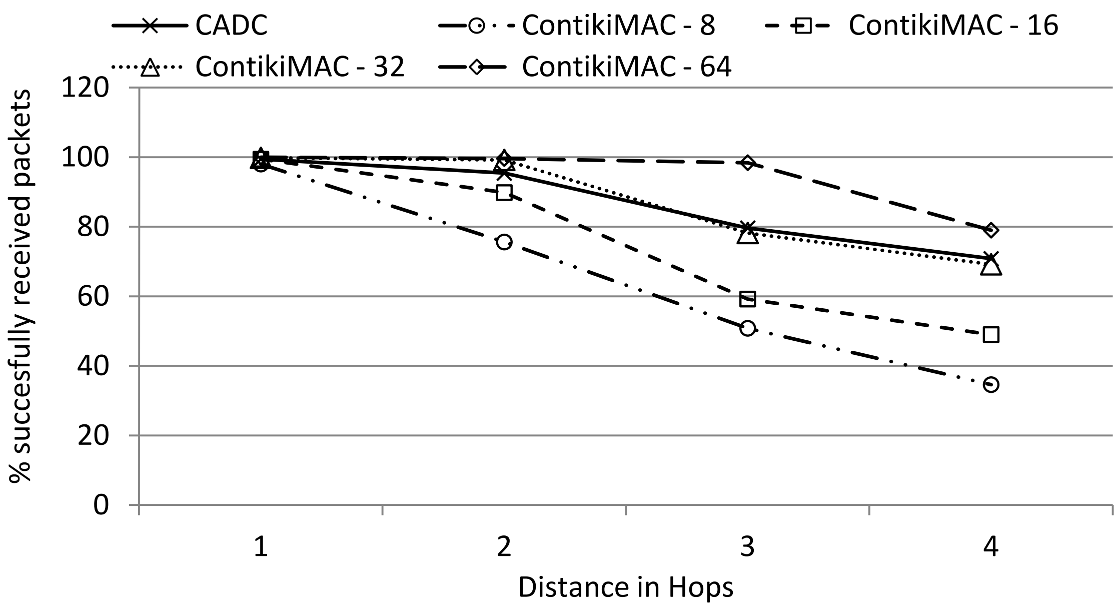

Figure 6.34: goodput for different channel check rates over distance in hops during high traffic configuration *(inter packet transmission interval 62.5 ms)*

traffic requirements. During our experiments, in order to achieve a valid comparison between the proposed protocol *CADC* and ContikiMAC, the range of values CADC could dynamically adjust its channel check rate was between 8 and 64 *(same parameter range as the different range of channel check rates used for ContikiMAC)*. Since CADC initially starts with the lowest channel check rate, some time is required for the recalculation and readjustment of the channel check rate. This process can lead to packet losses during high traffic scenarios and it is the main reason behind CADC's lower goodput compared to ContikiMAC with channel check rate of 64.

As mentioned previously and observed during simulations, CADC channel check rate adaptation requires some time and can lead to some packet losses during high traffic and thus congestion. Therefore, in CADC the highest ratio of packet losses occurs at the start and before CADC converges to the optimal channel check rate. Once CADC converges to the optimal channel check rate packet losses decrease and thus packet delivery ratio is also increased for the entire remainder of the traffic stream. In order to demonstrate this phenomenon, we also evaluated CADC under the same topology with varied number of packet transmissions. Figure 6.35 demonstrates the percentage of CADC's successfully received packets over different transmission lengths *(same interval and distance but different number of packets injected to the network)*. Further studying this figure, it can be observed that the percentage of successfully received packets was higher when the number of transmitted packets increased. This does not mean that CADC behaves differently for varied transmission lengths. On the contrary this result

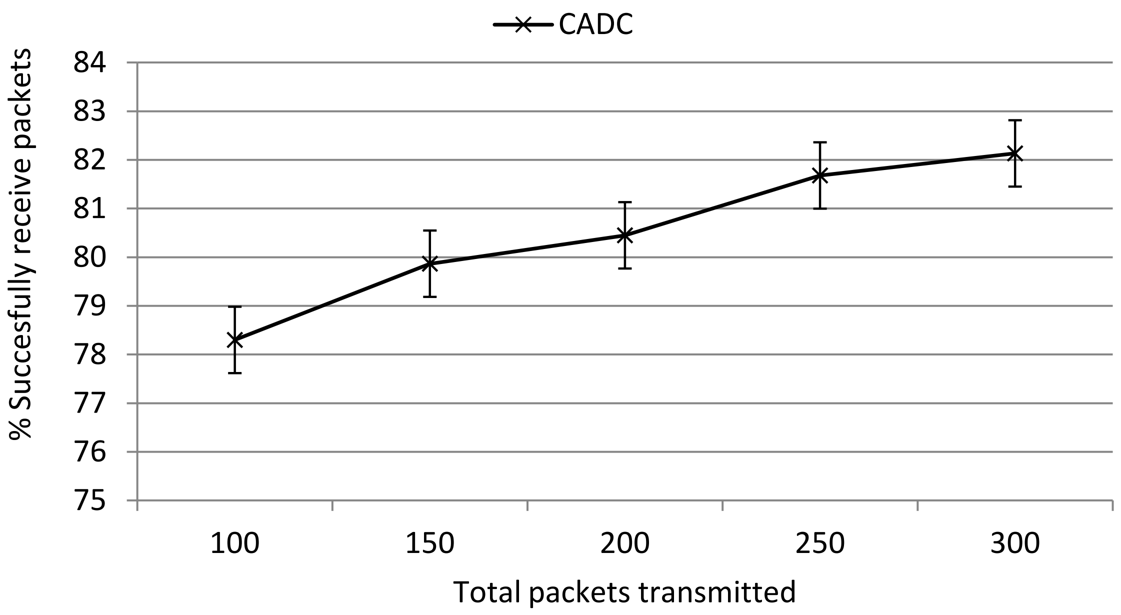

Figure 6.35: Percentage of successfully received packets over different transmission lengths *(Distance is three hops)*

demonstrate that CADC has a very stable behaviour by been able to identify the optimal channel check rate and adapt to it for as long as it is necessary. To further explain this it must be taken under consideration that part of CADC's dropped packets happened during the channel check adaptation phase. Since the channel check rate will be adapted once per flow *(if the traffic pattern does not change)* and the traffic is CBR during the experiments, the number of dropped packets due to the adaptation will remain the same for the varied length experiments. This in turn explains why CADC had higher packet reception ratio when we increased the number of packets for each transmission. CADC adaptation times are going to be discussed more detailed during subsubsection 6.6.2.3.

Overall CADC confronted congestion during various traffic patterns and successfully reconfigured nodes to the optimal channel check rates each time. Furthermore, CADC achieved highest goodput than the majority of ContikiMAC's channel check rate configurations. Moreover, CADC's goodput performance was comparable to ContikiMAC's highest channel check rate configuration.

6.6.2.2 Packet Loss

During test-bed experiments, traffic sources were CBR and UDP protocol was used for packet transmissions *(no packet retransmissions by the applications and thus duplicate packets).* Therefore, goodput represents the number of successfully received packets at the base station.Taking the above under consideration, packet losses can be calculated by subtracting goodput from the total number of packets transmitted.

Figure 6.33 illustrates goodput as % ratio of successfully received packets. From this figure, the %packet loss for each scheme can also be estimated as: % − goodput. Additionally, the length of each packet transmission in total packets transmitted is described in detailed during subsection 6.6.1 and thus more detailed information such as the precise number of packets lost can be easily estimated as:

$$\frac{\%packet\_loss \times transmission\_length}{100} \tag{6.4}$$

Further studying packet loss during test-bed experiments, it is shown that CADC achieved lower packet losses than the majority of ContikiMAC's channel check rate configurations. Furthermore, CADC's performance in terms of packet loss was comparable to ContikiMAC's highest channel check rate configuration *(ContikiMAC's highest channel check configuration demonstrated the lowest packet losses)*.

On the contrary to the simulations, during test-bed experiments packet loss during round-trip traffic scenarios was evaluated in order to enhance the analysis of CADC. For the measurement of round-trip packet losses ping6 with different transmission intervals was used. The choice of ping6's transmission intervals should represent both heavy and low traffic scenarios. Therefore two packet transmission intervals were used: 1 packet every second *(low traffic)* and 1 packet every 200ms *(heavy traffic)*. The ICMP payload in each packet was 64 bytes.

Figure 6.36 illustrates how packet loss increases for the different distance in hops during heavy and low traffic scenarios accordingly. Further studying the above figures shows that during high traffic, CADC always had some packet losses. As mentioned in the previous section, during high traffic packets can be lost during the channel check rate adaptation periods of CADC. The proposed protocol achieved no packet losses during low traffic, while it's packet loss during high traffic experiments was the second lowest after ContikiMAC's configuration with a CCR of 64. Overall, CADC demonstrated the same behaviour in both uni-directional and bi-directional traffic while it demonstrated a comparable to ContikiMAC's best configuration *(in terms of packet losses)* performance.

6.6.2.3 Packet Delay

On the contrary to simulators where usually most of the nodes have synchronised clocks, one of the most accurate ways to measure packet delay in testbeds with non synchronised clocks is by measuring the round trip time of a packet. In our testbed, "ping6" was used for the measurement of the round-trip times. Numerous pings were conducted from a linux PC connected to the WSN towards individual

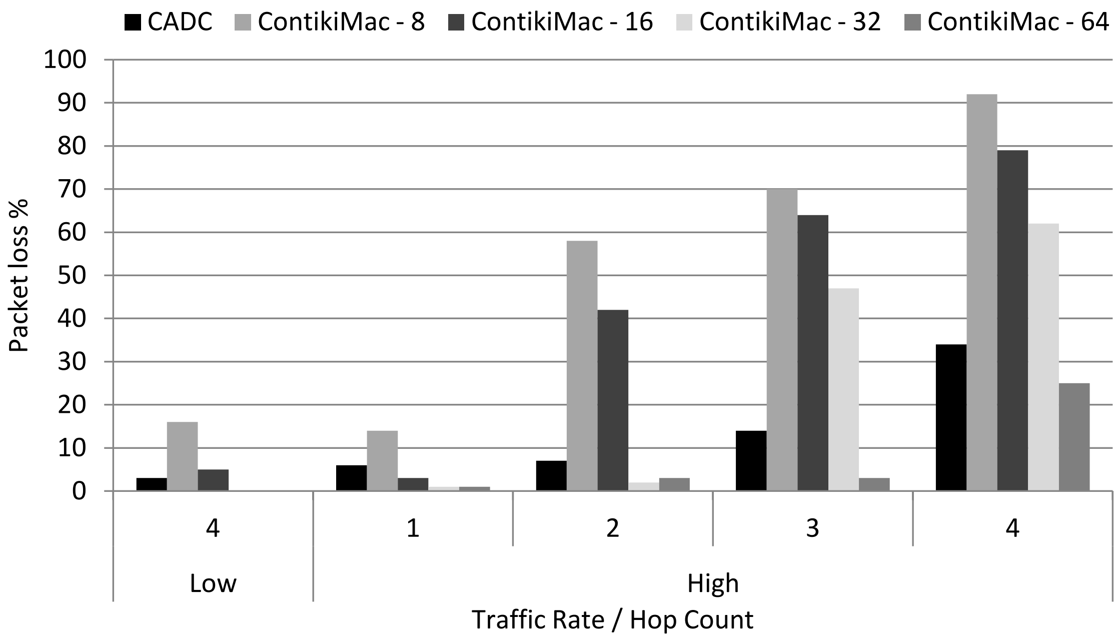

Figure 6.36: percentage of round trip packet loss during heavy and low traffic over different channel check rate and distance in hops *(categories with 0 loss are not presented for clarity)*

nodes. In order to show the round-trip delay in both heavy traffic and low traffic scenarios, two packet transmission intervals where used: 1 packet every second and 1 packet every 200ms. The ICMP payload for each packet was 64 bytes.

It is worthy to mention that packet intervals of 200 ms during pings cannot be directly compared to the packet intervals used for the goodput measurements since the traffic patterns are different *(ping is bi-directional traffic)*.

Figure 6.37 illustrates how distance in hops affects the round-trip packet delay during heavy and low traffic scenarios accordingly. By further studying the above figures, it can be observed that higher configurations of channel check rate for ContikiMAC exhibited lower delay times. Moreover, during the low traffic scenarios, the difference between the delay times between the different configurations was not as noticeable as in the high traffic experiments. During low traffic, different channel check configurations of ContikiMAC demonstrated small differences between the delay times. The largest difference observed in this experiments was between ContikiMac's channel check configurations of 64 and 8 in which there was an approximately 100% increase in the round-trip times. On the contrary, during the high traffic experiments, this difference increased to approximately 500%. This can be explained if we take under consideration that higher channel check rate configurations allow the nodes for a larger number of packet transmissions every second *(a detailed explanation of channel check rate and its relation to packet transmissions can be seen in section 6.3)*. This results in higher badnwidth for each link and thus packets queues build up slower.

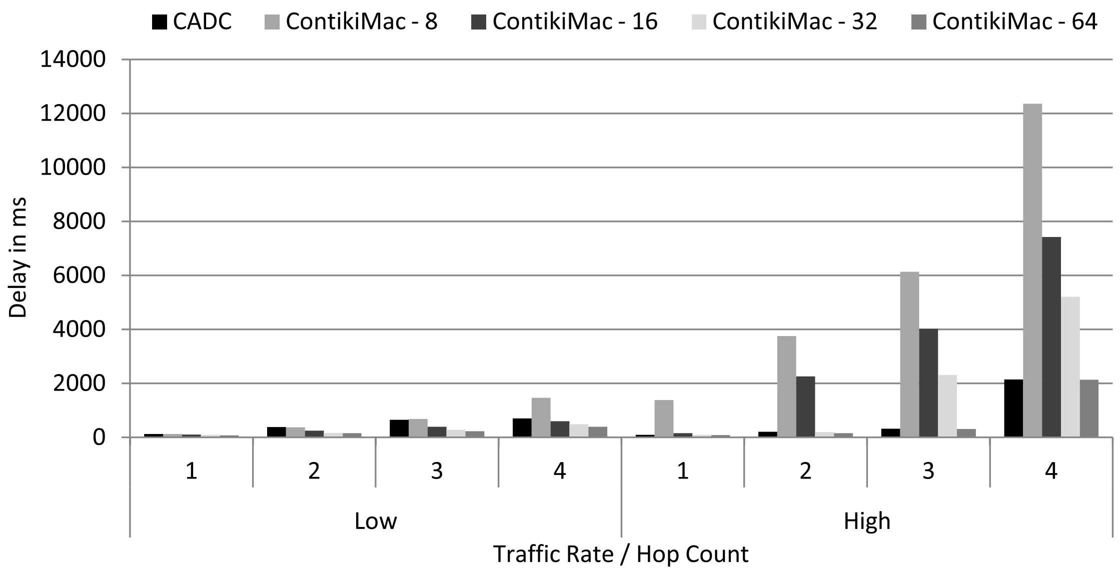

Figure 6.37: Round trip packet delay during heavy and low traffic over different channel check rate and distance in hops

In contrast to ContikiMAC, CADC demonstrated lower delay during experiments with high traffic load. Furthermore, the round-trip delay did not always increase between experiments with different distance between the source and the base station. Both of the above observations can be explained if we take under consideration the nature of the protocol. CADC adjust its channel check rate based on traffic requirements. Consequently, higher traffic in the network will result in a higher channel check rate adjustment which in turn is responsible for the lower delay times. Similarly to this, longer distances result in an increased number of transmissions per packet and thus higher interference. In the high traffic experiments, CADC successfully achieved the lowest possible round-trip delay similarly to the ContikiMAC's configuration with a channel check rate of 64. Additionally, during low traffic experiments, CADC achieved some of the lowest round-trip delay times.

As described in section 6.3, CADC does not require any node synchronisation and thus it can adapt to any traffic changes very fast. 6.38 shows in detail the per second delay of CADC in a 2 hop experiment. This figure illustrates how CADC reacts to high traffic conditions and how it adapts its channel check rate. During this experiment, it is observed that delay times kept increasing until CADC's congestion avoidance algorithm kicked in. After detecting congestion, CADC required approximately 2 seconds to initially adapt the channel check rates of the nodes. It is worthy to mention that since the traffic was bi-directional, every node in the traffic path would adapt its channel check rate and thus this is the highest possible adaptation time for a distance of two hops *(longer distances in hops would require higher adaptation times).*

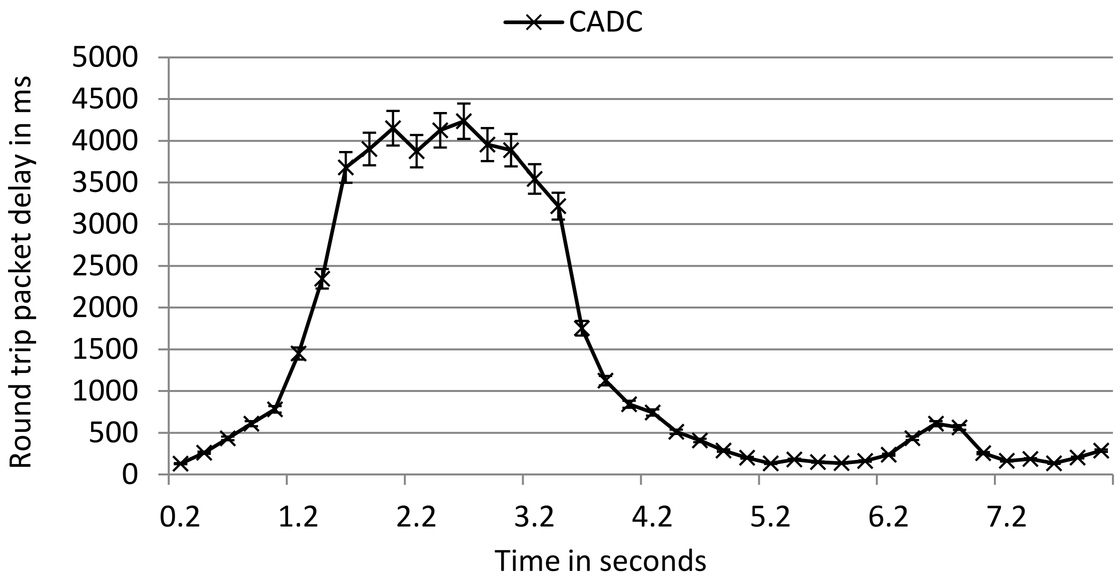

Figure 6.38: CADC round trip delay over time in seconds *(2 hops distance - 0.2 sec interval between consequent ping packets)*

Moreover, during high traffic and intense congestion CADC may perform more than one adaptations to the channel check rate which in turn will result in over all longer adaptation times. This can be justified if we further analyse a small increase in the delay times between 6.2 and 7.2 second of our experiment. The later inconsistency in the delay times was caused due to some changes in the configured channel check rates. CADC possibly identified that its current channel check rate configuration was higher than the required and decreased the channel check rate. This in turn led to some congestion *(channel check rate should be powers of 2. When the channel check rate is high 1 step changes may lead to significant changes in the performance)* and thus the channel check rate was again increased. If we further study this it is visible that this time it took only 0.5 second to adapt the channel check rates at the nodes.

6.6.2.4 Energy Consumption

In Figure 6.39, the per second energy consumption when the network is idle can be seen. In this figure it is visible that higher channel check rate configurations in ContikiMAC lead to higher energy consumption. When configured the network with ContikiMAC and a channel check rate of 64 had 3.35 times higher energy consumption from its channel check configuration of 8. Since channel check rate is pre-configured in ContikiMAC, lower cycle configurations can significantly increase the life-time of a WSN. When the network was idle CADC achieved the minimum posible energy consumption similarly to ContikiMAC's configuration with a channel check rate of 8. During these measurements, even though there

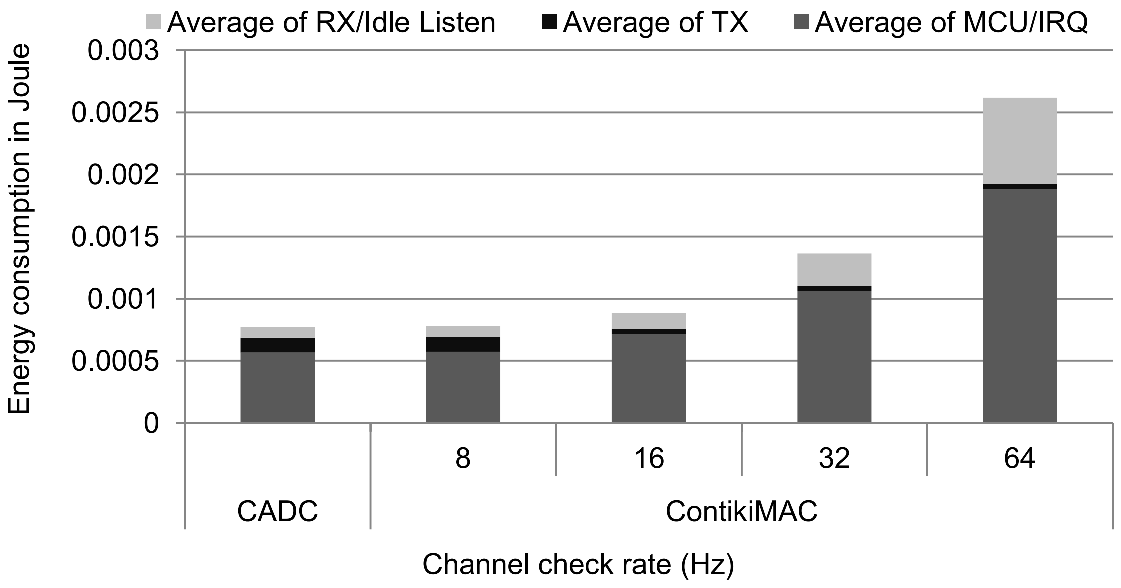

Figure 6.39: Per second energy consumption when network is idle

was no traffic generated from the nodes some packets such as RPL control messages and energy consumption measurment packets were still transmitted through the network. If there was no traffic at the network the above results would have shown an even larger difference in the energy consumption between the different permutations of ContikiMAC.

Figure 6.40 and Figure 6.41 shows the average energy consumption of the network for a 100 transmitted packets at the sources and the average energy consumption of the network per successfully received packet at the base station accordingly. Studying the above figures, it can be observed that higher channel check rates does not necessarily lead to higher energy consumption when the network is active. Moreover, when the same amount of packets injected in the network with different inter packet transmission intervals the energy consumption between the different permutations was significantly affected. These figures show that fixed duty cycle protocols with pre-configured cycle times can not achieve the best energy consumption when the traffic parameters and patterns vary. On the contrary, CADC achieved significantly lower energy consumption in every experiment. CADC's energy consumption was approximately half as much as ContikiMAC's lowest achieved energy consumption. This performance of CADC can be explained if we take under consideration that CADC nodes operate in the minimum energy consumption and only the nodes that participate in a traffic flow will increase their cycles. When the transmission ends CADC nodes will readjust their cycles to the minimum value. This mechanism of CADC incorporates both the advantages of low channel check rate when the network is idle and the advantages of higher channel check rates during various traffic rates.

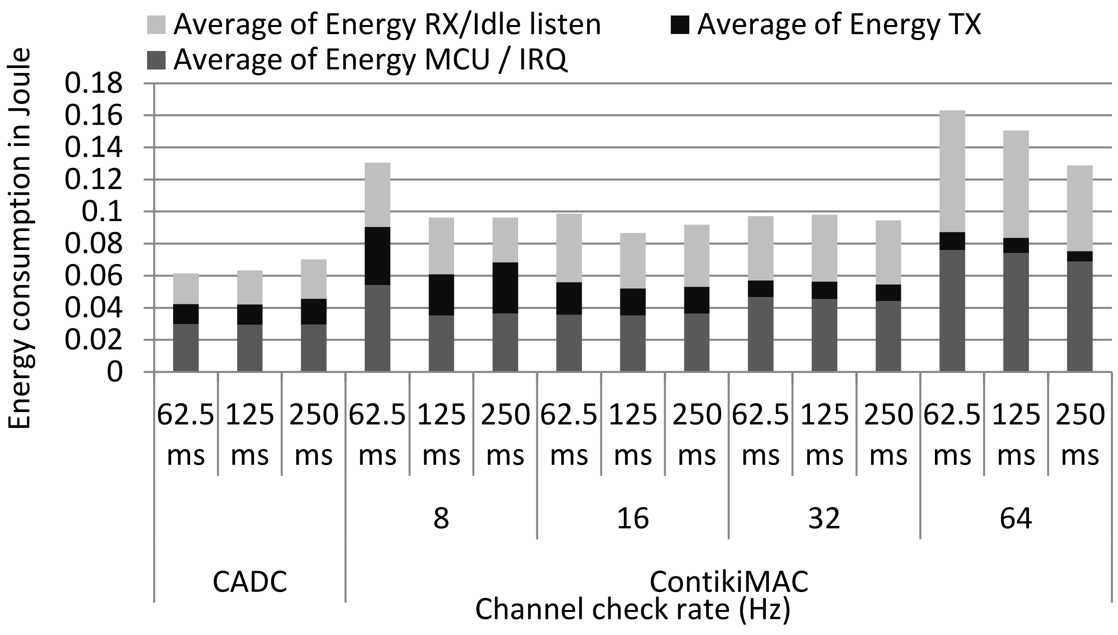

Figure 6.40: Average energy consumption for 100 transmitted from the sources packets

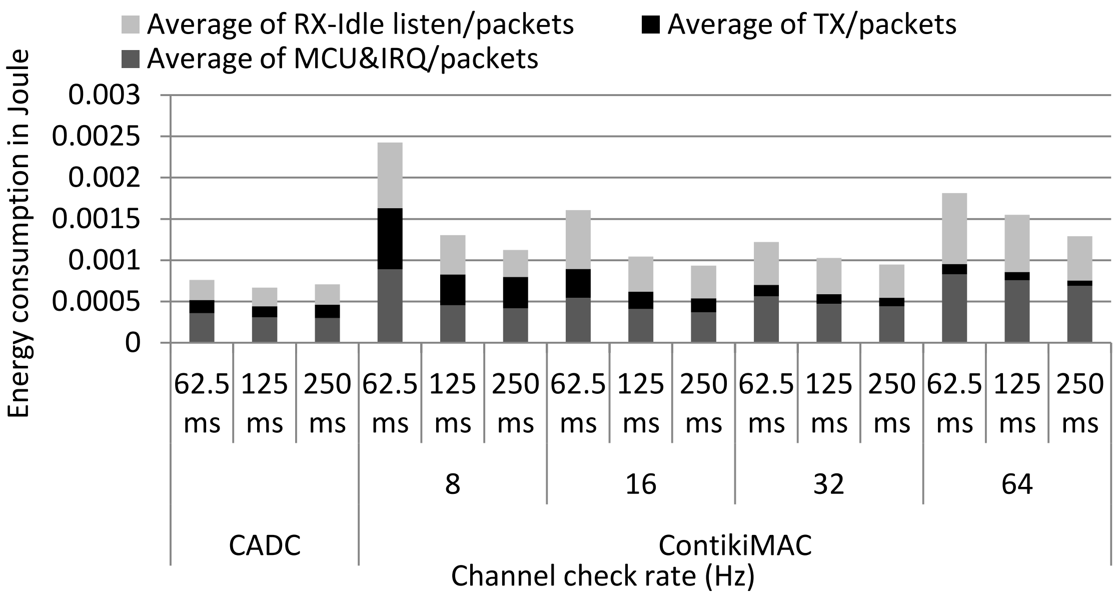

Figure 6.41: Average energy consumption per successfully received packet for 100 transmitted from the sources packets

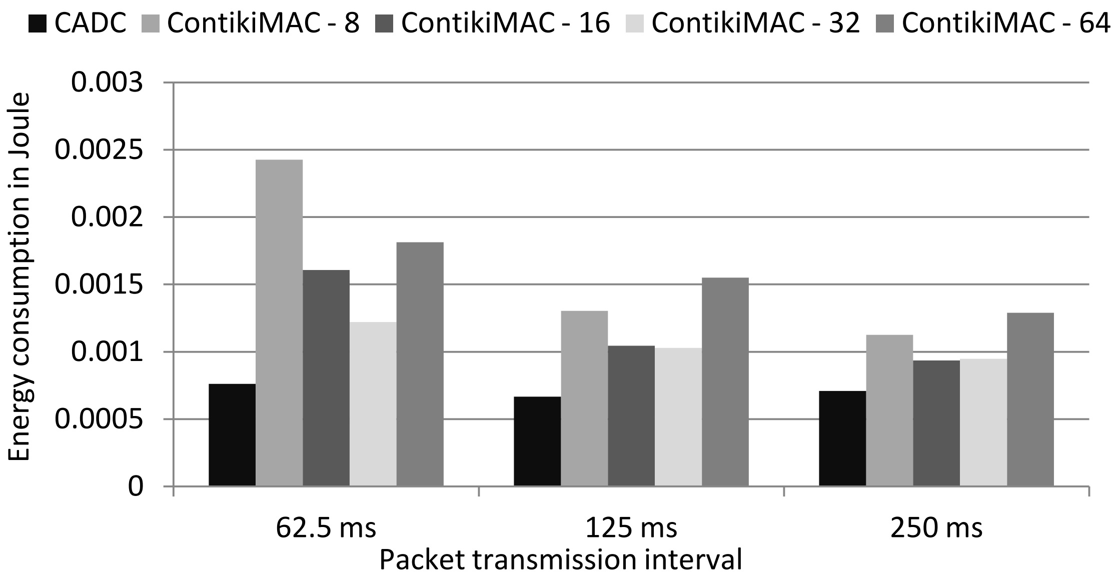

Figure 6.42: Network's, per packet average energy consumption over inter packet transmission interval

Figure 6.42 shows in detail how energy consumption is affected for the various packet transmission intervals. In the case of ContikiMAC, lower transmission rates *(larger intervals between packets)* resulted in lower energy consumption. Higher transmission rates have a higher probability for collisions since packets transmissions are closer to each other. Collisions in turn result in retransmission of packets and thus more energy is consumed. On the other hand, CADC demonstrated more stable energy consumption between the different transmission rates. By dynamically adjusting the channel check rates based on the traffic parameters, optimal energy consumption can be achieved and thus CADC can always deliver a close to ideal performance in terms of energy consumption.

Figure 6.43, illustrates energy consumption versus distance in hops. Both CADC and ContikiMAC had higher energy consumption when the distance between the source and the destination increased. This is expected since when the distance increases each packet requires more transmissions to reach the destination. It is worthy to mention that compared to CADC, ContikiMAC was much more affected when the distance increased. This phenomenon can be explained if we consider the connection between channel check rate and the duration of each packets transmission. As explained in section 6.2 when the channel check rate increases, packet train duration decreases. In CADC only the nodes in the traffic path adjust their channel check rates. Nodes outsides the path will keep operating in their default *(lowest)* channel check rate. This will result in partial incapability of packet hearing in nodes outside the traffic path since nodes with higher channel check rates will transmit packets for a smaller amount of time than the amount of time required by the nodes with lower channel check rates. Therefore non related to

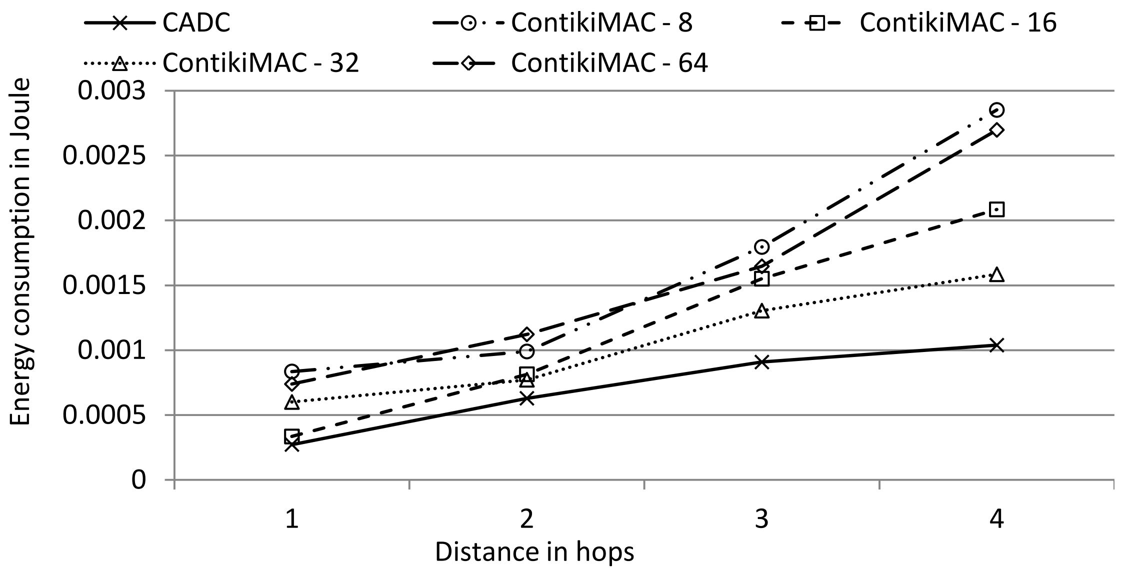

Figure 6.43: Network's, per packet average energy consumption over distance in hops

the traffic path nodes will not consume as much energy listening to packets not destined to them.

6.6.2.5 Memory Requirements

Table 6.4 shows the code and memory footprints for both protocols compiled with SDCC. The numbers in this table represent the combined RDC and MAC size for the two configurations under investigation. It can be observed that CADC requires approximately 2kb extra memory than the default Contiki configuration, which uses CSMA with ContikiMAC at the MAC and RDC layer respectively. This was expected since CADC incorporates many more functionalities in order to precisely calculate, adapt and co-ordinate cycle times.

Table 6.4: Protocol code and memory footprints in bytes using SDCC

	ContikiMAC + CSMA	CADC
In Flash		
CODE	7576	9426
CONST	58	58
In RAM		
XRAM	433	508
DATA	0	0

6.6.2.6 Test-bed Analysis

In the previous sections, the performance of a WSN configured with static DC protocol as well as with dynamic has been analysed in terms of goodput, energy consumption and delay. The above analysis has shown that the trade off between energy consumption and performance is significant when a static DC algorithm is used in the network. When the network is idle and configured with a static DC protocol *(ContikiMAC)*, there is a large diversity between the energy consumption of the various channel check rate configurations. During these experiments, when the network was configured with a channel check rate of 64, a 350% rise in the energy consumption was observed compared to its channel check rate 8 configuration. Based on that, it is easy to conclude that a primarily idle WSN can extend its life-span considerably when configured with lower channel check rate.

On the contrary, during network activity periods, the lowest channel check rate configuration did not achieve lower energy consumption. Studying Figure 6.40 and Figure 6.41 it is shown that for various transmission rates and distances in hops, different channel check rate configurations achieved the lowest energy consumption. As discussed in section 6.2, this observation can be explained if we consider that higher channel check rates lead to higher bandwidth. Based on the above, in order to achieve the best performance with the lowest energy consumption nodes should be frequently reconfigured based on traffic requirements. The decision of the optimal duty cycle for a sensor network can be even harder in 6LoWPAN where the communication patterns are arbitrary and multiple different applications may be operating at same WSN *(different packet sizes, bandwidth requirements and destination nodes)*.

The majority of the above problems can be solved if the network is configured to operate with a dynamic DC algorithm. Dynamic duty cycle protocols can adjust the network's channel check rate based on traffic requirements and therefore minimise energy consumption while maintaining high goodput and low delay times. In this study, as a solution to the above mentioned problems, we proposed CADC. Extensive test-bed experiments demonstrated that CADC can successfully adjust channel check rates, achieving the lowest energy consumption and round trip delay times while maintaining goodput at a level close to the maximum achievable.

6.7 Simulation Vs Testbed

In this research study, simulations aimed in the evaluation of the proposed schemes against other, previously designed implementations. The aim of test-bed experiments was the evaluation of the suggested schemes in real hardware and the confirmation of the simulation results. Generally, as described in section 3.3 hardware level simulated motes can produce more accurate results and informations about the low-level software such as device drivers. More specifically this type of COOJA motes uses exactly the same firmware that a real hardware device of the same type would. Therefore when this type of motes used for simulations in COOJA; results will be as close to a real test-bed as possible.

Due to various reasons such as radio interference *(during test-bed experiments, the interference was significantly higher)*, signal reflection and others *(discussed in detail in section 3.10)*, simulation experiments cannot perfectly achieve identical to test-bed experiments results. A significant reason behind this is the different hardware used for the test-bed and for the simulation experiments *(hardware simulated)*. Additionally, in our case except the environmental conditions, there are some additional characteristics related to hardware restrictions affecting the test-bed results. These characteristics are mainly code optimisations and modifications for the functionality of Contiki OS and the proposed algorithms in Sensinode devices. These code optimisations are summarised in the remainder of this section.

By default, the official port of Contiki for devices with the cc24xx MCU and thus our Sensinode devices include some stack optimisations such as reduction of the maximum possible stack depth by reducing the maximum number of callback functions *(callback functions in Contiki OS can also be referred as blocking callbacks. This type of callback function is called before the end of the execution of the current function)*. Detailed information about the default CC24xx optimisations in Contiki can be found in [145].

Due to stack overflows and memory constrains, ContikiMAC the default duty cycle protocol in Contiki OS is not functional in the official Contiki port for Sensinode devices. Duty cycle is one of the most important functionalities in a self powered WSN. Additionally, a big part of this research is based on how duty cycle affects the operation of WSN and thus congestion control algorithms designed for non duty cycling WSN. Therefore additional optimisations and modifications were made to Contiki as part of this research study in order to achieve a fully functional, duty cycling WSN with our Sensinode devices. These code optimisations include:

- Conversion of variables to Static type. In 8051 MCU the stack is independent and it is 256 bytes. Static variables does not get stored in the stack while Local variables do. Therefore significant amount of bytes can be prevented from getting in the stack.

- Contiki has numerous timer modules. Some of them can be used in interrupt contexts *(r-timer)*. R-timer is a hardware timer and used by contiki's duty cycle algorithms for timing precision *(through interrupts DC functions can be called exactly when the timer expires)*. Since the stack in Sensinode devices is limited, such an approach can result in stack overflows which in turn leads to node crashes *(Stack is already very high when the timer expires and the interrupt calls other functions related to duty cycle)*. In order to prevent that, a stack control check has been added to the r-timers *(interupt)* callback function. When the timers expired and the interrupt function is called, the stack control check is performed. If the stack does not have enough space to acomodate the duty cycle routines *(node will crash)*, the procedure will be delayed for μSec till there is enough space in the stack.

- Contiki's clock module is platform dependent and is implemented in the file clock.c. Since the clock module handles the system time, the clock module implementation usually also handles the notifications to the etimer library when it is time to check for expired event timers. Instead of checking the expired event from withing the clocks interrupt function, a flag is set. Expired event timers are handled by the main function. This way no unexpected callbacks can occur and thus no stack overflows.

Detailed information and actual codes related to Contikis stack optimisations and modifications can be found in Appendix D.

Furthermore, CADC was configured with a minimum channel check rate of four during the simulation experiments. Based on the simulation results, when the network was configured with a channel check rate of four the results was not significantly contributing any additional results of importance for the evaluation of CADC. Therefore, CADC was configured with a minimum channel check rate of eight during the test-bed experiments. This in turn can result in lower converge times during test-bed experiments. As shown in Figure 6.38, CADC has very fast adaptation times and thus the differences in the performance of CADC due to different minimum channel check rate configurations will not lead to inconsistency or noticeable differences in the performance.

Except the above mentioned environmental, hardware and configuration differences; the design of test-bed experiments was different from the simulations. Parameters such as source nodes, destination, transmission rate and number of packets to be transmitted by the sources were pre-configured during simulations whereas in real test-bed experiments nodes could dynamically receive the configuration for each packet transmission. In reality, exactly the same as the one used during test-bed experiments could be used during simulations *(Network simulator communicate with the base station through a border router node)*. A simulation design like this have significant draw-backs. Firstly, based on the load of simulated nodes, COOJA will operate in different non-real simulation speed. For example COOJA simulator may need 10 real life seconds in order to simulate 1 second of experiment. Taking this under consideration, real-time communication between the simulator and the real PC would be almost impossible. Therefore, there is a very high risk of inaccurate measurements and thus results. Additionally, a simulation design similar to test-bed would render simulation automation impossible *(a user must select the active configuration before each experiment)* and thus gathering the same amount of experimental data would require many additional months of experiments. As a conclusion, it is not wise to use the exact same network design in both simulations and test-bed experiments.

Even though there are notable differences between simulations and test-bed, the parameters important for our experiments remained the same in both simulations and real hardware experiments *(algorithms, transmission rates, duty cycle values, distance in hops, etcetera)*. Since the aim of this study is not the evaluation of COOJA simulator, the implementation methods used and the experiments conducted are sufficient for the evaluation of CADC and the rest of the duty cycle schemes.

6.7.1 Goodput

Figure 6.44 demonstrate CADC's goodput in an easy to compare simulation and testbed format. All the non common information between test-bed and simulations have been removed from these figures. By comparing this two figures it is visible that overall CADC demonstrated the same behaviour between simulation and testbed experiments. The testbed experiments performed after the simulations. Through the simulations, it was observed that some configurations such as packet transmission intervals above 250 ms and channel check rate below 8 are not contributing any extra -significant results to the experiments. Therefore, in order to reduce the number of configuration permutations during the test-bed experiments and focus on the more significant results, these configurations were excluded.

During simulations, CADC's minimum channel check rate configuration was set to 4 while the same value was set to 8 during the test-bed experiments. Having a lower threshold for the minimum channel check rate, in some cases it may result in more CADC state changes till the scheme settles in the optimal channel check rate. This in turn will result in slightly lower goodput. On the contrary, the results obtained through simulations was slightly better for CADC. Additionally, during test-bed experiments, it was observed that distance in hops had a greater impact to both CADC's and ContikiMAC's goodput. When the distance was two hops, both simulation and test-bed demonstrated similar goodput. On the other hand, when the distance increased to four hops a performance gap is visible between hardware and simulation results. More specifically, during the test-bed experiments, when the distance was four hops and the inter packet transmission rate at the sources was 62.5 ms, CADC had 70% of the transmitted packets succesfully trceived while ContikiMAC configured with a channel check rate of 64 had 78%. With the same network configuration, during simulations CADC had 85% of the transmitted packets succesfully trceived while ContikiMAC achieved 100%. Similar to test-bed's loss for the distance of 4 hops is observed for distances of 5 - 6 hops during simulations.

There are various reasons behind these differences between simulation and testbed. One of the most important reasons is the environment. In simulation experiments the radio is configured to operate without any "outside" interference *(from factors that are not related to the experiment).* On the contrary, during test bed experiments and especially when the test-bed is set up indoors, the "outside" interference can be unpredictable and affect the results. Moreover, even though the same scenarios were tested in both simulations and test-bed, the network topology was not the same. As demonstrated previously, line topology simulation experiments included nodes in a line and each node was neighbour only with its parent and child nodes *(low interference).* During test-bed experiments the topology was consisted by 15 nodes randomly deployed and mostly in range of each other. Therefore, during test bed experiments the interference was significantly higher when the distance increased in hops. In order to further explain this it is necessary to understand that even though the distance was the same in hops, the network density and thus the radio interference was significantly higher. For example, when the distance was 4 hops each node could interfere only with 1 other node *(it's parent node transmission)* during the simulations but this number was increased to 3 at the testbed experiments *(all the nodes between the source and the destination).* Additionally, for the measurement of energy consumption additional traffic had to be transmitted through the network. In some cases this additional traffic could affect the performance of the schemes in terms of goodput *(not always possible to perfectly gather the goodput results during the idle time between the energy measurement periods).*

Additionally, during line topology simulations each node had only one parent node. Therefore the networks topology could not change. On the contrary, during test-bed experiments nodes had multiple neighbour nodes and thus routing options. Taking under consideration the unstable nature of RPL, routing paths could change dynamically during the experiments *(the longest the path between the source and the destination, the more this phenomenon can be observed).* This in turn could affect the test-bed experiments since switching traffic paths may result in some packet losses or even temporary unavailability.

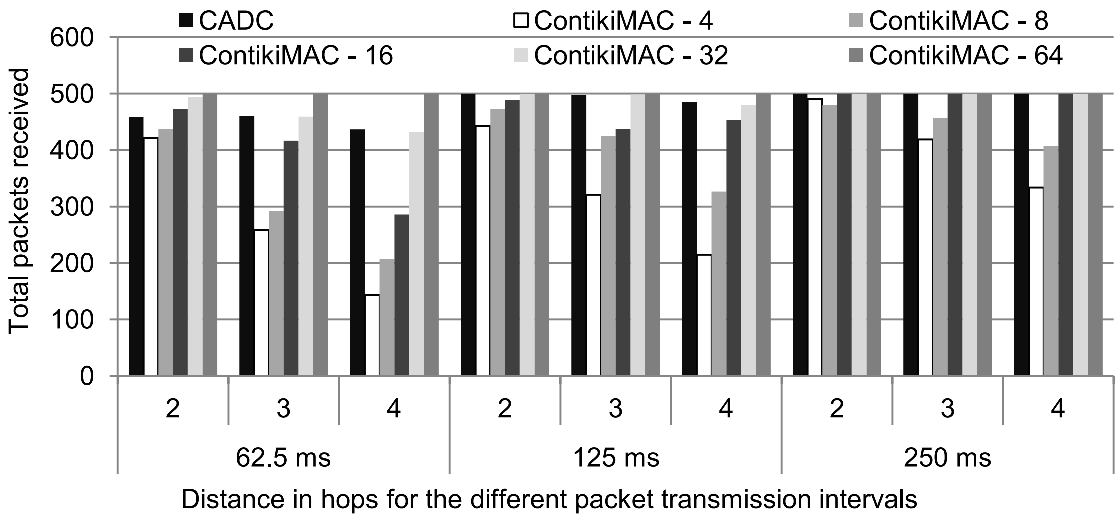

Figure 6.44: Goodput during simulations *(line topology)*

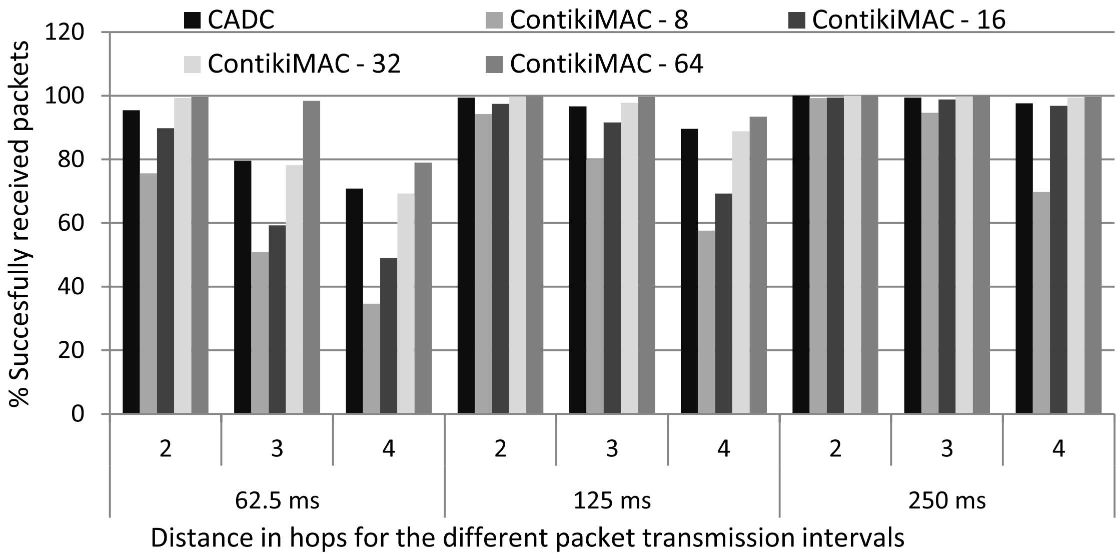

Figure 6.45: Goodput during test-bed experiments

Except the above mentioned environmental and design differences between simulations and test-bed, there were hardware and OS differences as well. The simulated motes were tmotesky while the ones used during the test-bed were Sensinode. Different hardware can behave differently even when the same codes are executed. Some examples of the differences between the simulated and the real nodes were discussed previously *(in section 3.6 discussed the different energy consumptions while different random number generators where discussed in section 5.6)*.

Finally the code optimisations for the Sensinode devices could have some impact on the goodput. For example, both the modifications in Contiki's clock and in Contiki's r-timer could result in delayed *(μSec)* packet receptions. If we further analyse this stack optimisations, the stack control check modifications in r-timers interrupt function could even lead to packet losses in some cases. To explain this in detail, lets assume the duty cycle's algorithm timer expires and the radio must turn on to scan for incoming packet transmissions. If the stack is too high, turning on the radio will be delayed by μSec. Even though such a small delays would not affect the packet reception in reality, if the stack levels happens to be high consequently; it is possible that the node may skip a whole cycle *(this is possible only for high CCR configurations such as 64, where cycles can be as small as 15ms)*.

6.7.2 Packet Loss

Since the total number of transmitted packets was the same between the experiments and the traffic sources were UDP, it is easy to conclude that the schemes with the highest goodput had also the lowest packet loss. For the majority of the experiments, the packet loss performance of the schemes is linked with their performance in terms of goodput.

Similarly to the goodput analysis, CADC achieved the second lowest packet loss after ContikiMAC's configuration with a channel check rate of 64. Additionally, it is worthy to mention that distance in hops had a greater impact to both CADC's and ContikiMAC's packet loss. When the distance was two hops, both simulation and test-bed demonstrated similar goodput. On the other hand, when the distance increased to four hops a performance gap is visible between hardware and simulation results. More specifically, during the test-bed experiments, when the distance was four hops and the inter packet transmission rate at the sources was 62.5 ms, CADC suffered an approximately 30% packet loss while ContikiMAC configured with a channel check rate of 64 suffered an aproximately 22% loss. With the same network configuration, during simulations CADC's packet loss was only 15% while ContikiMAC's *(64 CCR)* loss was 0. This high loss is observed for distances of 5 - 6 hops during simulations.

The reasons behind this are: the environment, the topology, the starting channel check rate and the code optimisations used during the test-bed experiments *(more details can be found in subsection 6.7.1)*.

In order to verify how much the various traffic patterns affect the schemes; additional experiments with bi-directional traffic have been conducted. In this experiments, ping6 was sued for the generation of bi-directional traffic. Moreover, different ping6 intervals have been tested in order to generate various traffic rates.

During these experiments, Both CADC and ContikiMAC demonstrated the same behaviour as the one with uni-directional traffic. CADC in most cases achieved the second lowest loss after ContikiMAC *(CCR 64)* during high traffic experiments. During high traffic, it is note worthy that for the distances of 1 and 2 hops ContikiMAC's configuration of 32 channel check rate achieved lower loss than CADC as well. The main reason behind this lies in CADC's adjustment to the optimal channel check rate. When the distance is only 1 or 2 hops the interference is lower and thus CADC does not need to adjust it's channel check rate to a very high value. More specifically, through CADC's results it is highly possible that the algorithm did not settle in one channel check rate when the distance was 2 hops. For example, CADC originally increased the channel check rate to 32 but this channel check rate was judged as an over-configuration by the algorithm. Therefore CADC decreased the channel check rate to 16 but this configuration was low for the load of traffic in the network and thus the algorithm increased again to 32 in order to drain the queues. Since the channel check rate can be only powers of 2, this behaviour is expected. The current duty cycle decrease algorithm of CADC is configured to operate in this manner in order to minimise energy consumption *(bouncing between two channel check rates when needed)*. On the other hand this may cost some goodput by causing some additional packet losses. If it is required from the networks design, CADC can be configured with a less sensitive duty cycle reduction algorithm *(reduce only when absolutely needed)* and thus prevent scenarios like the above mentioned.

6.7.3 Packet Delay

During simulations, packet delay was measured as the average time needed by each packet to reach the destination. In our testbed, "ping6" was used for the measurement of the round-trip times. Even though both CADC and ContikiMAC are designed to operate with various traffic patterns *(uni-directional / bi-directional)*, a direct comparison between the one-way delay times during the simulations and the round-trip times during test-bed cannot be made. This is because protocols may operate slightly different during bi-directional traffic and the interference at the nodes is much higher *(each packet is transmitted two times by each node)*.

Even though, the main objective of the test-bed experiments was validate the performance of the protocols in a real environment. Figure 6.46 illustrates the packet delay during simulations filtered with parameters similar to the one's used during the test-bed experiments.

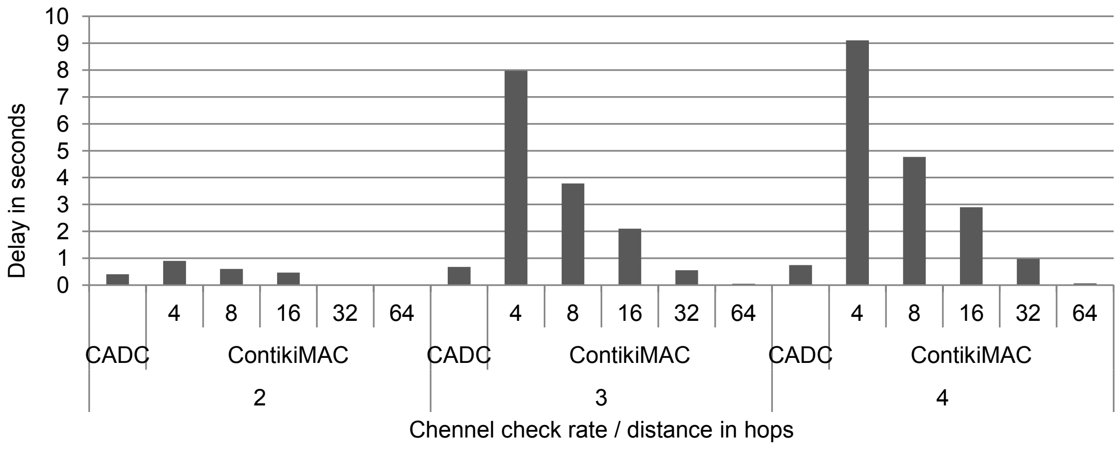

Figure 6.46: Packet delay for different distance in hops with various channel check rate configurations *(simulation)*

Comparing the above figure with Figure 6.37 *(testbed delay)*, it can be observed that when the distance increased delay increased as well. During test-bed experiments, ContikiMAC demonstrated a similar to the simulations performance. It is worthy to mention that a very high rise in the delay times is observed, when the distance increased to four hops in test-bed experiments. The reasons behind this are the same as the reasons described in section 6.7): the environment, the topology, the starting channel check rate and the code optimisations used during the test-bed experiments.

On the contrary to the simulation experiments where CADC had the second lowest delay after ContikiMAC's configuration with a channel check rate of 64; CADC demonstrated one of the lowest delay during test-bed experiments *(hi traffic)*, similar to ContikiMAC's highest configuration. Studying Figure 6.1 this decrease in CADC's delay times can be explained. During simulations experiments, traffic was uni-directionsl. This in turn may lead to different channel check configurations between the nodes and thus higher delay times. To be more precise, some of the nodes may have lower channel check rates and thus transmit less packets per second. Therefore the packet queues at these nodes may built up which in turn increases the delay times. On the contrary, During test-bed experiments the traffic was bi-directional. This led to the same channel check configuration at every node in the traffic path and thus lower delay times.

The above described behaviour of CADC can be distinguished easier during low traffic scenarios *(Figure 6.37)*. When the traffic was low, it can be observed that CADC demonstrated delay times similar to other -lower channel check configurations of ContikiMAC *(CADC's optimal channel check rate was not the highest configuration during that scenarios)*.

Overall, simulation and test-bed experiments demonstrated that both CADC and ContikiMAC can successfully operate with low delay times in both uni and bi directional traffic scenarios.

6.7.4 Energy Consumption

6.7.4.1 When Network is Idle

Figure 6.24 and Figure 6.39 illustrate the energy consumption when the network is idle during simulations and test-bed experiments accordingly. Studying the above figures, it can be observed that Sensinode *(used for test-bed experiments)* had much higher energy consumption in every network configuration. It is also notable that the energy consumption due to radio TX, RX/idle listen is similar in both test-bed and simulation results. On the contrary the MCU related energy consumption is on a totally different scale between test-bed and simulation experiments. To be more precise the MCU energy consumption was 116 times higher during test-bed experiments. This difference is mainly caused by the different hardware used during simulations and test-bed experiments. Table 6.5 demonstrates the energy consumption for the different platforms. Studying this table, it can be observed that the energy consumption for the different radio states is similar in both Tmotesky and Sensinode. On the other hand, the energy consumption for the different MCU states is by far greater in Sensinode.

Additionally, during test-bed experiments, nodes were transmitting packet periodically in order to gather the information related with the energy consumption at each node *(more details can be found in section 3.11)*. The presence of these packets in the network can also be a reason for small variations in the energy consumption between test-bed and simulation experiments.

Running experiments with different sensor nodes and using periodical packets for energy measurement can justify the difference in energy consumption between test-bed and simulation results.

Table 6.5: Energy consumption for the different platmforms

	Tmotesky *(simulations)* Joule/sec	Sensinode *(test-bed)* Joule/sec
TX	0.04959	0.0582
RX/Idle listen	0.056145	0.0576 *(MCU/IRQ)*
MCU	0.00156*(MCU-hi)*	0.0171
	0.00000812*(MCU-LPM)*	

Even though there were noticeable differences in energy consumption, both

ContikiMAC and CADC demonstrated the same behaviour and energy consumption patterns between simulations and test-bed experiments. CADC achieved the best performance by having the minimum possible energy consumption *(same as ContikiMAC's configuration with the lowest channel check rate).*

6.7.4.2 Network is Active

In order to achieve a more detailed comparison between test-bed and simulations Figure 6.47 illustrates only the common with the test-bed data, of how distance in hops affects energy consumption. Comparing the above figure with Figure 6.43 *(same experiments test-bed)*, it can be observed that during simulations, distance in hops affected energy consumption much less than in test-bed experiments. This performance difference related to distance in hops; is caused due to the environment, topology and the code optimisations used during the test-bed experiments *(more details can be found in subsection 6.7.1).*

Moreover, a big difference can be observed in the performance of ContikiMAC configured with a channel check rate of 64. Furthermore, it is observed that the energy consumption per successfully received packet was higher in the simulations. Based on the energy consumption difference between Sensinode and Tmotesky, simulation should have had lower energy consumption per successfully received packet compared to their corresponding configuration on test-bed. The cause of this observation can also be blamed for ContikiMAC's *(with channel check rate of 64)* noticeable performance difference.

During test-bed experiments, energy consumption was measured periodically. Usually, packet transmissions were happening between the periodic energy measurements. As a consequence, during test-bed experiments, only the network activity periods were measured. On the contrary, simulations were configured to last 10 minutes *(in order to achieve automation).* This in turn resulted in measuring a significant amount of idle time in every simulation experiment *(this amount of idle time was larger for higher channel check rate configurations).* Therefore, energy measurement data for the simulations include mixed data *(idle time and high network activity)* which in turn is responsible for the two abnormalities described in the previous paragraph. Other less significant factors contributing to the energy variation between simulation and testbed experiments include:

- During test-bed experiments, MCU had significantly higher energy consumption. This in turn contributes to a less visible energy consumption difference between the various configurations of ContikiMAC.

- When the stack is in dangerous levels *(possibility of stack overflow and thus node crash)*, turning on the radio will be slightly delayed. This phenomenon

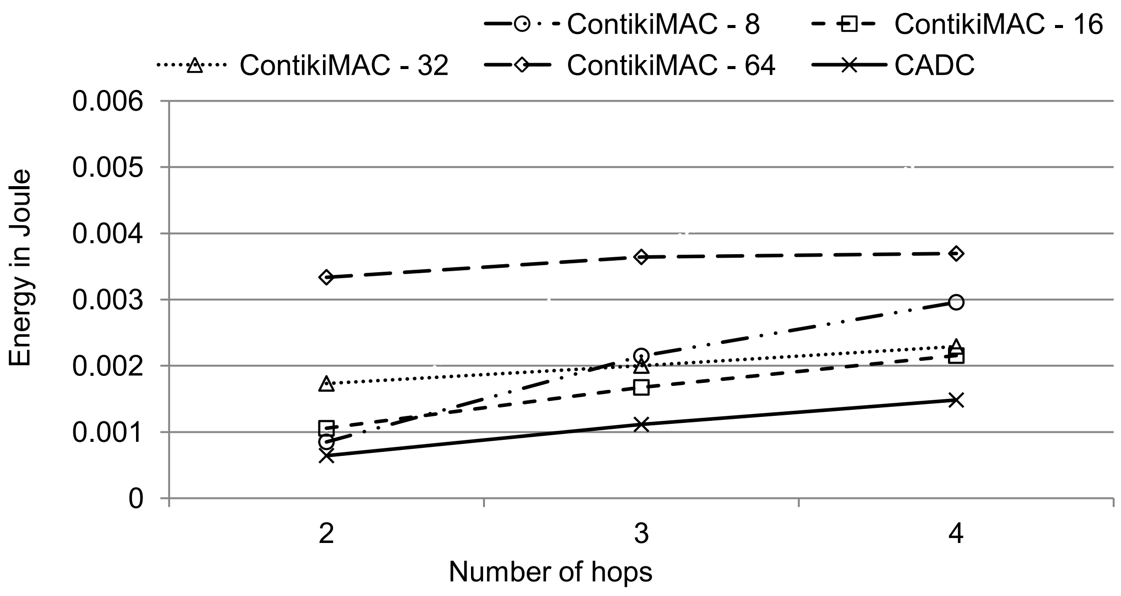

Figure 6.47: Per packet energy consumption over distance in hops *(simulation)*

is usually more frequent during high channel check rate configurations and consequent occurrences thereof may result in missing a whole cycle.

It is worthy to mention that the above parameters affecting the energy consumption measurements apply to all energy measurements and thus similar differences in the performance can be observed in every energy consumption comparison between simulations and test-bed.

Overall, both ContikiMAC and CADC demonstrated the same behaviour patterns in both simulations and test-bed experiments. In both cases when the distance in hops increased so did the energy consumption while larger intervals between packet transmissions resulted in lower energy consumption. Furthermore, lower channel check rates do not result in lower energy consumption when the network is active. CADC achieved the lowest energy consumption and the most stable behaviour in every network configuration *(compared to every channel check rate configuration of ContikiMAC)*.

6.7.5 Memory Requirements

Analysing Table 6.2 and Table 6.4, it can be observed that CADC was larger than ContikiMAC with CSMA by 900 bytes for the Tmotesky platform while the difference increased to 2kb when the Sensinode platform was used. In general, the size of the executables produced for the Sensinode platform was significantly larger *(5.3 KB more for ContikiMAC/CSMA and 6.4 KB more for CADC)*. This can explain why the difference, in terms of memory, between ContikiMAC and CADC increased for the Sensinode platform.

There are two main reasons leading to the significantly larger code size for the Sensinode platform.

1. The compiler used by Sensinode *(for 8051 architectures)* is SDCC. On the contrary, Tmotesky use MSP430-GCC for the compilation of the codes. Different compilers can perform their compilation tasks slightly differently and thus various compilers have a different degree of code optimisation. This can result in various executable sizes for the same code.

2. Sensinode have a 8051 MCU which is 8 bit architecture. Compared to this Tmotesky use a 16 bit MCU. Joseph Yiu and Andrew Frame investigated if 8bit, 16bit or 32bit MCUs are the best option for designs requiring small program code size in [151]. This study shown that the 32bit architecture had considerably smaller code size than both the 16bit and 8bit architectures. More precisely, the code size of 8bit architectures was significantly larger while 16bit architecture produced a code size close to that of the 32bit architecture. In reality, although some 8bit processor instructions are one byte in length, many instructions are actually two bytes or even three bytes long.

   In most applications it is necessary to process 16bit or larger data. For example, in an 8bit architecture, the integer data type is 16bit. Therefore, every time an integer or a C library function that supports an integer is used, 16bit data is being processed. This processing requires a long sequence of instructions for an 8bit architecture.

6.7.6 Limitations

A great number of simulation and test-bed experiments have been conducted in this thesis for the evaluation of CADC. In both simulation and test-bed experiments, CADC demonstrated similar behavior and it is demonstrated that it can significantly improve the performance of the WSN and reduce the energy consumption. Some limitations in the evaluation of CADC follow:

1. **Topologies.** In this thesis, two topologies were used for the evaluation of CADC *Line and random topology*. Therefore, CADC has not been evaluated under other WSN topologies. CADC nodes operate independently without the need of synchronisation. Additionally, CADC propagates congestion information in a hop-by-hop fashion by encapsulating the information in the header of each out-going packet. Moreover, WSN traffic patterns are significantly more limited than traditional networks. Taking the above under consideration, it can be concluded that for the majority of WSN topologies,

the performance of CADC will not be significantly affected. More experiments and detailed analysis of CADC under different network topologies will be performed in the future.

2. **Source traffic.** In the majority of the experiments sources were transmitting in a constant bit rate *(CBR)*. Therefore, the behavior of CADC has not been evaluated under sources transmitting in varied bit rates. CADC's design, is focused on fast adaptations to traffic changes. CADC nodes does not need synchronisation or any other time consuming process before duty cycle adaptations. Figure 6.38 demonstrates CADC's adaptation times. Even though CADC is designed for traffic changes and VBR traffic rates will not significantly affect its behaviour, we can predict that its packet losses due to cycle adaptation times will increase. Experiments studying the behavior of CADC in VBR environments will be conducted as part of the future work.

3. **CADC CCR limitations.** During both simulation and testbed experiments, CADC was configured with a maximum and a minimum CCR. The maximum CCR during the experiments was 64. This can explain why during this research study we always compared CADC's performance with the performance of ContikiMAC configured with a CCR of 64. In the majority of the experiments, CADC demonstrated a similar performance *(in terms of goodput)* with ContikiMAC with a 64 CCR. In these cases, CADC achieved the best possible good put performance since its maximum CCR configuration was 64. If CADC's upper CCR limit was set to a higher value, it's goodput would have possibly surpassed ContikiMAC with a CCR of 64. On the other hand this would have been an unfair comparison between a dynamic and a static MAC and thus it was not evaluated in this thesis.

4. **Behaviour of the sink.** During simulation and testbed experiments, it was observed that when the traffic is uni-directional, the sink will not trigger the duty cycle reduction algorithm. This is basically happening due to CADC's nature. CADC measure the traffic parameters based on packet transmissions. Since sink will never transmit a packet in a uni-directional traffic scenario, it will never enter the over duty cycle state and thus reduce its CCR. This will result in sink nodes reducing their CCR only when they become inactive. This does not have a big impact on CADC's performance since nodes with higher CCR can always receive packet from nodes with lower CCR. Additionally, the sink usually operates as a gateway node *(connects WSN with the traditional networks)* and thus it is always connec-

ted to a power source *(connected to a PC in order to forward the traffic to the traditional network)*. This can easily be overcome by having the sink update CCR based on a timer or the received transmissions instead of the transmitted packets.

6.8 Summary and Discussions

Through simulations and test-bed experiments, this chapter has detailed analyse the impact of duty cycles on the performance of WSNs. Based on the experimental results, higher cycles will result in higher energy consumption when the network is idle. This is not always the case during network activity. Based on the load of traffic in the network, the different duty cycles demonstrated varied performance in terms of goodput, packet loss and energy consumption. Therefore, in order to achieve the optimal performance; a WSN must be reconfigured *(reprogrammed)* whenever the traffic characteristics change. Frequent reprogramming in large scale sensor network implementations is difficult and energy consuming *(in case over the air reprogramming)*. The decision of the optimal duty cycle for a sensor network can be even harder in 6LoWPAN where the communication patterns are arbitrary and multiple different applications may be operating at same WSN *(different packet sizes, bandwidth requirements and destination nodes)*.

With the use of dynamic duty cycle schemes, the majority of the above problems can be solved. Dynamic duty cycle protocols can adjust the network's channel check rate based on traffic requirements and therefore minimise energy consumption while maintaining high goodput and low delay times. In this study, as a solution to the above mentioned problems, we proposed CADC. CADC is designed as a standalone MAC layer, and thus it is easy to combine with different routing protocols under various topologies. Therefore, CADC is independent of network topology, routing protocol. Furthermore, CADC is protocol independent and can be used with various WSN applications with different traffic requirements. Additionally, CADC can successfully detect and confront congestion while it forward's packets based on their priority. Moreover, CADC is the first dynamic duty cycle protocol implemented for Contiki OS as well as one of the first schemes designed based on the arbitrary traffic characteristics of IPv6 WSN. Even though CADC is designed for IPv6 and 6LoWPAN networks, been designed as a stand alone MAC scheme and can easily be transferred to any WSN architecture *(various protocol stacks)*.

Over 10000 simulations experiments and 700 test-bed experiments have been conducted for the evaluation of CADC. Through these experiments, it is demonstrated that CADC successfully adapted its cycle based on traffic patterns in every traffic scenario. Furthermore, CADC does not require any time synchronisation algorithms to operate at the nodes and does not use any additional packets for the exchange of information between the nodes *(no overhead).* Overall, CADC achieved the lowest energy consumption, very low packet delay times and packet loss. Moreover, Results indicate that CADC's goodput was better than other dynamic duty cycle schemes and comparable to the highest goodput observed among the static duty cycle configurations.

Finally, CADC constitutes a very stable new proposal for the dynamical adjustment of duty cycles at the nodes. In the future this work can be used as a reference for future MAC protocol architecture aiming in high performance with low energy consumption algorithms.

Chapter 7

Conclusions and Future Work

This thesis focuses on research into congestion control in duty cycle 6LoWPAN networks. In this chapter, a summary of the completed work is given in section 7.1 and potential future work is described in section 7.2.

7.1 Conclusions

Wireless sensor Networks *(WSN)* have evolved from the idea that small wireless sensor devices can be used for the collection of information from the physical environment. A wireless sensor deployment usually consists of spatially distributed autonomous sensors monitoring their surrounding environment. Collected data traverse the network towards sink nodes in a multi-hop fashion. Modern sensor networks usually make use of bi-directional communication links which in turn allows additional control of the sensors activity. Bi-directional links with the ability to access nodes individually are usually achieved through sensor architectures such as IPv6 over Low power Wireless Personal Area Networks *(6LoWPAN)* as each node has its own IP address. This has led to the increased popularity of IPv6 and related specifications by the WSN community. In general, congestion occurs when offered traffic load exceeds available capacity to the degree that quality of service deteriorates. In WSN, congestion can cause a plethora of malfunctions such as packet loss, increased delays, lower throughput and energy inefficiency, potentially resulting in a reduced deployment lifetime and under-performing applications. Furthermore, it has been shown that idle radio listening is a major source of energy consumption and thus low-power wireless devices must keep their radio transceivers off for energy saving. This has driven the WSN community into developing power saving MAC protocols with Radio Duty Cycling (RDC).

In literature there is a great number of congestion control schemes. However, careful study of previous work reveals that RDC schemes are often neglected during the design and evaluation of congestion control algorithms. Additionally, most of the existing congestion control schemes are tailored to specific routing patterns and thus are not suitable for 6LoWPAN networks. In order to demonstrate the importance of duty cycles, chapter 4 contains an evaluation of previously suggested congestion mechanisms in IPv6 and 6LoWPAN network configured with and without duty cycle. To address the problem of congestion, two novel congestion control schemes have been proposed in this thesis:

1. DCCC6 a new Duty Cycle-Aware Congestion Control scheme for 6LoWPAN Networks. DCCC6 performs congestion detection based on a dynamic buffer. When congestion is detected, parent nodes will inform the nodes contributing to congestion resulting in a local rate adaptation and thus congestion mitigation. The rate adaptation algorithm implemented by DCCC6 aims for local fairness between neighbour nodes. The child notification procedure will be decided by DCCC6 and will be different when the nodes are duty cycling. When the nodes are duty cycling the child notification will be made through unicast frames. On the contrary broadcast frames will be used for congestion notification when the network is not duty cycling.

   The performance analysis is presented in chapter 5. Both simulations and test-bed experiments have been conducted for the evaluation of DCCC6. Additionally, experiments dedicated to the comparison between simulation and test-bed experiments have been carried out. Results analysis has shown that DCCC6's performance was better than previous works in terms of goodput and packet loss. DCCC6 also achieved the lowest energy consumption per successfully transmitted packet. Moreover, the overall energy consumption, delay and fairness levels were also competitive when compared to other schemes.

2. CADC a new Congestion Aware Duty Cycle MAC. In order to achieve fast adaptation times with precision, CADC measures congestion levels based on node activity. The congestion level sampling rate is higher when traffic increases. When nodes are idle, there is no possibility of congestion occurring and thus sampling stops. CADC nodes, measure their queue occupancy periodically *(every few packet transmissions)*. As well as that, nodes will keep records and monitor the status of packet transmissions in order to identify the cause of congestion. Two queue thresholds “low” and “high” are implemented at each CADC node. Nodes are considered to be in a light congested state if queue occupancy exceeds the low queue threshold and in heavy congestion state when queue occupancy exceeds the high threshold. Additionally, a transmission threshold is applied to the measured packet

statistics in order to identify the cause of congestion. When a node increases its channel check rate, layer 2 frame transmission rate will also be increased. This in turn can contribute to continuous congestion even after a channel check rate increase *(a scenario capable of causing a phenomenon like the above mentioned can be: adjustment of the channel check rate for both children and parent nodes at the same time)*. In order to prevent this, nodes also measure the number of incoming packets in addition to queue occupancy and packet transmission status ratios. The aforementioned metrics are then combined in order to reconfigure node channel check rates.

CADC's dynamic CCR adaptation is designed to achieve energy efficiency, high performance, scalability and very fast adaptiion times without the need of a clock synchronisation algorithm between nodes. Moreover, CADC is independent of network topology, routing protocol and the application at each node. Furthermore, CADC is the first dynamic duty cycle protocol designed based on the arbitrary traffic characteristics of 6LoWPAN. Even though CADC is designed for IPv6 and 6LoWPAN networks, it is designed as a stand alone MAC scheme and can easily be transfered to any WSN architecture *(various protocol stacks)*.

The performance analysis of CADC contains over 10000 simulations experiments and 700 test-bed experiments. Through these experiments, it is demonstrated that CADC successfully adapted its cycle under any network configuration tested. Overall, CADC achieved the lowest energy consumption, lowest packet loss and very competitive delay times, compared to both other dynamic duty cycle schemes and static duty cycle configurations. Furthermore, results indicate that CADC achieved a higher goodput than other dynamic duty cycle schemes while it can be compared to the highest goodput observed among the static duty cycle configurations.

The schemes proposed in this research study, are the first congestion control work ever implemented in Contiki OS *(at the time of writing)*. Contiki OS has seen an immense growth during the last years and thus our work is an important contribution to the Contiki OS research community and an example for future network mechanism architects. 6LoWPAN is believed to be the future of WSN *(connecting every object with the Internet: "The Internet of Things")*. Since the majority of the implemented congestion control schemes are not tailored based on the unique characteristics of IPv6 and 6LoWPAN networks, there is a great need for the development of congestion control mechanisms designed for this type of networks. The proposed schemes are some of the first works that consider the existence of duty cycle in the network and are tailored for the unique characteristics of IPv6

6LoWPAN networks. Therefore, the proposed schemes are a valuable contribution to the WSN research community and a significant step towards the development of duty cycle, IPv6 ready and congestion/traffic aware MAC protocols.

7.2 Future Work

Recommendations for future work are presented here based on the discussions from the previous chapters and illustrations of potential applications of DCCC6 and CADC in a wider aspect.

- To investigate the performance of DCCC6 and CADC with QoS, energy and congestion aware routing protocols.
- To assess the performance of DCCC6 and CADC when there is node mobility in the network and extend the proposed schemes accordingly.
- Investigate the performance of DCCC6 and CADC with different sensor node hardware, such as platforms that COOJA can simulate and conduct a detailed comparison of COOJA simulator and related real hardware test-bed.
- To explore the feasibility of combining CADC with DCCC6 or another MAC layer rate adaptation scheme.
- To extend DCCC6 and CADC schemes to include packet priority transmission mechanisms. A weight function will be applied to each packet and thus packets will not be transmitted in FIFO order, but based on their priority.
- To explore the feasibility of dynamic transmission synchronisation in CADC. Nodes should keep records of their parent nodes previous wake up times and avoid unnecessary packet transmissions at the RDC layer *(similar scheme is implemented in ContikiMAC).*
- To further investigate the possibility of extending CADC with a MAC recognition mechanism. This mechanism should be able to detect the underlying MAC/RDC layer used by neighbour nodes *(based on the transmission patterns or use of reserved bits in 802.15.4 header).* This can result in a dual MAC layer configuration in the network and possibly better performance. Self powered nodes will operate with CADC for energy saving while the rest of the nodes can operate without duty cycle *(e.g. nullRDC or an extended version of CADC)* in order to increased bandwidth and reduce the collision occurrence *(packet train transmissions significantly contribute to collision occurrence).*

References

[1] J. Leslie (IETF 77), "What do you mean, "congestion"?" Anaheim, CA, USA, Tech. Rep., March 2010. [Online]. Available: http://www.ietf.org/proceedings/77/slides/iccrg-7.pdf

[2] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Proceedings of Real-Time in Sweden 2007*, Vsters, Sweden, Aug. 2007.

[3] Wikipedia, "IEEE 802.15.4," retrieved Jun 2012. [Online]. Available: http://en.wikipedia.org/wiki/IEEE_802.15.4

[4] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," Swedish Institute of Computer Science, Tech. Rep. T2011:13, Dec. 2011.

[5] J. P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP*. Morgan Kaufmann, 2010.

[6] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537 – 568, 2009.

[7] J. W. Ye, Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2002, pp. 1567 – 1576 vol.3.

[8] M. Buettner, G. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, ser. SenSys '06. New York, NY, USA: ACM, 2006, pp. 307–320.

[9] R. Musaloiu, C. M. Liang, and A. Terzis, "Koala: Ultra-low power data retrieval in wireless sensor networks," in *Proceedings of the 7th international conference on Information processing in sensor networks*, ser. IPSN '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 421–432.

[10] S. H. Lee, J. H. Park, and L. Choi, "Amac: Traffic-adaptive sensor network mac protocol through variable duty-cycle operations," in *Communications, 2007. ICC '07. IEEE International Conference on*, june 2007, pp. 3259 – 3264.

[11] J. Jeon, J. W. Lee, J. Y. Ha, and W. H. Kwon, "Dca: Duty-cycle adaptation algorithm for ieee 802.15.4 beacon-enabled networks," in *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, april 2007, pp. 110 –113.

[12] S. Bac, D. Kwak, and C. Kim, "Information networking. towards ubiquitous networking and services," T. Vazão, M. M. Freire, and I. Chong, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Traffic-Aware MAC Protocol Using Adaptive Duty Cycle for Wireless Sensor Networks, pp. 142–150.

[13] N. Saxena, A. Roy, and J. Shin, "Dynamic duty cycle and adaptive contention window based qos-mac protocol for wireless multimedia sensor networks," *Computer Networks*, vol. 52, no. 13, pp. 2532 – 2542, 2008, research and Trials for Reliable VoIP Applications Wireless Multimedia Sensor Networks.

[14] H. Chen, G. Yao, and H. Liu, "Traffic adaptive duty cycle mac protocol for wireless sensor networks," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, oct. 2008, pp. 1 –4.

[15] R. Alberola and D. Pesch, "Dcla: A duty-cycle learning algorithm for ieee 802.15.4 beacon-enabled wsns." in *ADHOCNETS'10*, 2010, pp. 217–232.

[16] D. Lee and K. Chung, "Adaptive duty-cycle based congestion control for home automation networks," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 1, pp. 42 –47, february 2010.

[17] M. Anwander, G. Wagenknecht, T. Braun, and K. Dolfus, "Beam: A burst-aware energy-efficient adaptive mac protocol for wireless sensor networks," in *Networked Sensing Systems (INSS), 2010 Seventh International Conference on Network Sensing Systems*, june 2010, pp. 195 –202.

[18] IEEE Computer Society, "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) Networks (WPANs)," The Institute of Electrical and Electronics Engineers, Inc., Tech. Rep., 2003.

[19] ——, “Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) – Amendment 1: Add Alternate PHYs,” The Institute of Electrical and Electronics Engineers, Inc., Tech. Rep., 2007.

[20] , “Approved draft amendment to ieee standard for information technology-telecommunications and information exchange between systems-part 15.4:wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans): Amendment to add alternate phy (amendment of ieee std 802.15.4),” *IEEE Approved Std P802.15.4a/D7, Jan 2007*, 2007.

[21] IEEE Computer Society, “Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) Networks (WPANs) Amendment 2: Alternative Physical Layer Extension to support one or more of the Chinese 314316 MHz, 430434 MHz, and 779787 MHz bands,” The Institute of Electrical and Electronics Engineers, Inc., Tech. Rep., 2009.

[22] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, *Transmission of IPv6 packets over ieee802.15.4 networks*, RFC 4944, 2004.

[23] N. Kushalnagar and G. Montenegro and C. Shumacher, *IPv6 over lowpower wireless personal A rea networks (6lowpans)*, RFC 4919, 2007.

[24] Swedish Institute of Computer Science (SICS), “The Contiki OS,” retrieved Jun 2012. [Online]. Available: http://www.contiki-os.org/p/about-contiki.html

[25] Wikipedia, “Contiki,” retrieved Jun 2012. [Online]. Available: http://en.wikipedia.org/wiki/Contiki

[26] Swedish Institute of Computer Science (SICS), “Contiki FAQ,” retrieved Jun 2012. [Online]. Available: http://www.sics.se/contiki/wiki/index.php/FAQ

[27] H. Ritter, J. Schiller, T. Voigt, A. Dunkels, and J. Alonso, “Experimental Evaluation of Lifetime Bounds for Wireless Sensor Networks,” in *Proceedings of the Second European Workshop on Sensor Networks (EWSN2005)*, Istanbul, Turkey, Jan. 2005. [Online]. Available: http://www.sics.se/~adam/ewsn2005.pdf

[28] J. Titus, “6lowpan goes where zigbee can’t,” Tech. Rep., December 2009. [Online]. Available: http://www.ecnmag.com/articles/2009/02/6lowpan-goes-where-zigbee-cant

[29] Zach Shelby, "Zigbee vs. ipv6?" retrieved Jun 2012. [Online]. Available: http://zachshelby.org/2009/02/20/zigbee-vs-ipv6/

[30] EEtimes, "Ipv4, ipv6, the internet of things, 6lowpan, and lots of other stuff," retrieved Jun 2012. [Online]. Available: http://www.eetimes.com/electronics-blogs/maxs-cool-beans-blog/4217647/IPv4--IPv6--The-Internet-of-Things--6LoWPAN--and-lots-of-other--Stuff-

[31] K. Ashton, "That 'internet of things' thing," *RFID Journal*, July 2009.

[32] P. Magrassi and T. Berg, "A world of smart objects," *Gartner research report*, August 2002.

[33] D. Uckelmann, M. A. Isenberg, M. Teucke, H. Halfar, and B. S. Reiter, "An integrative approach on autonomous control and the internet of things," *Gartner research report*, 2010.

[34] M. Kranz, P. Holleis, and A. Schmidt, "Embedded interaction: Interacting with the internet of things," *Internet Computing, IEEE*, vol. 14, no. 2, pp. 46 –53, march-april 2010.

[35] S. Bae, D. Kim, M. Ha, and S. H. Kim, "Browsing architecture with presentation metadata for the internet of things," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, dec. 2011, pp. 721 –728.

[36] A. Iera, C. Floerkemeier, J. Mitsugi, and G. Morabito, "The internet of things [guest editorial]," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 8 –9, december 2010.

[37] T. Stathopoulos, R. Kapur, D. Estrin, J. Heidemann, and Z. Lixia, "Application-based collision avoidance in wireless sensor networks," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, nov. 2004, pp. 506 – 514.

[38] I. Chatzigiannakis, A. Kinalis, and S. Nikoletseas, "Wireless sensor networks protocols for efficient collision avoidance in multi-path data propagation," in *Proceedings of the 1st ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, ser. PE-WASUN '04. New York, NY, USA: ACM, 2004, pp. 8–16.

[39] Y. Wang and J. J. Garcia-Luna-Aceves, "A hybrid collision avoidance scheme for ad hoc networks," *Wirel. Netw.*, vol. 10, pp. 439–446, July 2004.

[40] K. Jamieson, B. Hull, A. Miu, and H. Balakrishnan, "Understanding the real-world performance of carrier sense," in *Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, ser. E-WIND '05. New York, NY, USA: ACM, 2005, pp. 52–57.

[41] M. Heusse, F. Rousseau, R. Guillier, and A. Duda, "Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless lans," in *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '05. New York, NY, USA: ACM, 2005, pp. 121–132.

[42] S. Eisenman and A. Campbell, "E-csma: Supporting enhanced csma performance in experimental sensor networks using per-neighbor transmission probability thresholds," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May 2007, pp. 1208 –1216.

[43] H. C. Le, H. Guyennet, and N. Zerhouni, "A new contention access method for collision avoidance in wireless sensor networks," in *Networking, 2007. ICN '07. Sixth International Conference on*, april 2007, p. 27.

[44] J. I. Choi, M. Jain, M. A. Kazandjieva, and P. Levis, "Inverting wireless collision avoidance," Tech. Rep., 2009. [Online]. Available: http://sing.stanford.edu/pubs/sing-09-00.pdf

[45] A. Samanta and D. Bakshi, "Fault tolerant wireless sensor mac protocol for efficient collision avoidance," *CoRR*, vol. abs/1006.3369, 2010.

[46] Y. Sankarasubramaniam, O. B. Akan, and I. F. Akyildiz, " ESRT: Eventto-Sink Reliable Transport in Wireless Sensor Networks," in *Proc. 4th ACM international symposium on Mobile ad hoc networking & computing* , 2003.

[47] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," in *Proc. 2nd international conference on Embedded networked sensor systems* , 2004.

[48] C. T. Ee and R. Bajcsy, "Congestion control and fairness for many-to-one routing in sensor networks," in *In ACM SenSys.* ACM Press, 2004, pp. 148–161.

[49] H. Zhang, A. Arora, Y. Choi, and M. G. Gouda, "Reliable bursty convergecast in wireless sensor networks," in *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, ser.

MobiHoc '05. New York, NY, USA: ACM, 2005, pp. 266–276. [Online]. Available: http://doi.acm.org/10.1145/1062689.1062724

[50] Y. G. Iyer, S. Gandham, and S. Venkatesan, "Stcp: a generic transport layer protocol for wireless sensor networks," in *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on*, 2005, pp. 449–454.

[51] C. Wang, K. Sohraby, and B. Li, "SenTCP: A Hop-by-Hop Congestion Control Protocol for Wireless Sensor Networks," in *IEEE INFOCOM 2005 (Poster Paper*, 2005.

[52] S. Chen and Z. Zhang, "Localized algorithm for aggregate fairness in wireless sensor networks," in *Proc. International Conference on Mobile Computing and Networking Proceedings*, 2006.

[53] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis, "Interference-aware fair rate control in wireless sensor networks," in *Proc. ACM SIGCOMM Computer Communication Review*, 2006.

[54] S. Chen and N. Yang, "Congestion avoidance based on lightweight buffer management in sensor networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 9, pp. 934 –946, sept. 2006.

[55] C. Wang, K. Sohraby, V. Lawrence, B. Li, and Y. Hu, "Priority-based congestion control in wireless sensor networks," in *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (SUTC'06) - Volume 01*, ser. SUTC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 22–31.

[56] M. Zawodniok and S. Jagannathan, "Predictive congestion control protocol for wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 6, no. 11, Nov. 2007.

[57] J. Paek and R. Govindan, "RCRT: rate-controlled reliable transport for wireless sensor networks," in *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2007, pp. 305–319.

[58] H. tae Kim, R. Sankar, J. sik Lee, and I. ho Ra, "Hop-by-hop based reliable congestion. control protocol for wireless sensor networks," Tech. Rep., 2007. [Online]. Available: isis2007.fuzzy.or.kr/submission/upload/A1100.pdf

[59] Y. Ouyang, F. Ren, C. Lin, T. He, C. Li, Y. Hu, and H. Wen, "A simple active congestion control in wireless sensor network," in *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, oct. 2007, pp. 1 –7.

[60] J. P. Sheu, L. J. Chang, and W. K. Hu, "Hybrid congestion control protocol in wireless sensor networks," in *Proc. Vehicular Technology Conference*, 2008.

[61] K. Karenos, V. Kalogeraki, and S. V. Krishnamurthy, "Cluster-based congestion control for sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, no. 1, pp. 5:1–5:39, Feb. 2008.

[62] E. Giancoli, F. Jabour, and A. Pedroza, "Collaborative transport control protocol for sensor networks," in *Computer and Electrical Engineering, 2008. ICCEE 2008. International Conference on*, dec. 2008, pp. 373 –377.

[63] M. H. Yaghmaee and D. Adjeroh, "A new priority based congestion control protocol for wireless multimedia sensor networks," in *Proceedings of the 2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks*, ser. WOWMOM '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1–8.

[64] X. Yin, X. Zhou, R. Huang, and Y. Fang, "A fairness-aware congestion control scheme in wireless sensor networks," *IEEE Trans. Veh. Technol.*, vol. 58, no. 9, Nov. 2009.

[65] C. Wang, B. Li, K. Sohraby, M. Daneshmand, and Y. Hu, "Upstream congestion control in wireless sensor networks through cross-layer optimization," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 4, pp. 786 –795, may 2007.

[66] S. Misra, V. Tiwari, and M. S. Obaidat, "Lacas: learning automata-based congestion avoidance scheme for healthcare wireless sensor networks," *IEEE J.Sel. A. Commun.*, vol. 27, no. 4, pp. 466–479, May 2009.

[67] Y. M. Liu and X. H. Jiang, "A extended dccp congestion control in wireless sensor networks," in *Intelligent Systems and Applications, 2009. ISA 2009. International Workshop on*, may 2009, pp. 1 –4.

[68] W. W. Fang, J. M. Chen, L. shu, T.S.Chu, and D. P. Qian, "Congestion avoidance detection and alleviation in wireless sensor networks," in *Joural of Zhejiang University*, 2009.

[69] P. Antoniou and A. Pitsillides, "A bio-inspired approach for streaming applications in wireless sensor networks based on the lotka-volterra competition model," *Comput. Commun.*, vol. 33, no. 17, pp. 2039–2047, Nov. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2010.07.020

[70] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 1, pp. 2 – 16, feb 2003.

[71] J. Kang, B. Nath, Y. Zhang, and S. Yu, "Adaptive resource control scheme to alleviate congestion in sensor networks," in *the First Workshop on Broadband Advanced Sensor Networks*, 2004.

[72] C. yih Wan and A. T. Campbell, "Siphon: Overload traffic management using multi-radio virtual sinks in sensor networks," in *in SenSys*, 2005, pp. 116–129.

[73] L. Popa, C. Raiciu, I. Stoica, and D. Rosenblum, "Reducing congestion effects in wireless networks by multipath routing," in *Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, ser. ICNP '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 96–105.

[74] K. Jaewon, Z. Yanyong, and B. Nath, "Tara: Topology-aware resource adaptation to alleviate congestion in sensor networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 7, pp. 919 –931, july 2007.

[75] C. Sergiou, V. Vassiliou, and A. Pitsillides, "Reliable data transmission in event-based sensor networks during overload situation," in *Proceedings of the 3rd international conference on Wireless internet*, ser. WICON '07. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, pp. 31:1–31:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1460047.1460086

[76] M. Rahman, M. Monowar, and S. H. Choong, "A qos adaptive congestion control in wireless sensor network," in *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*, vol. 2, feb. 2008, pp. 941 –946.

[77] M. Ahmad and D. Turgut, "Congestion avoidance and fairness in wireless sensor networks," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, 30 2008-dec. 4 2008, pp. 1 –6.

[78] J. M. Huang, C. Y. Li, and K. H. Chen, "Talonet: A power-efficient grid-based congestion avoidance scheme using multi-detouring technique in wireless sensor networks," in *Wireless Telecommunications Symposium, 2009. WTS 2009*, april 2009, pp. 1 –6.

[79] O. Banimelhem and S. Khasawneh, in *Telecommunications, 2009. ICT '09. International Conference on, title=Grid-based multi-path with congestion avoidance routing (GMCAR) protocol for wireless sensor networks*, may 2009, pp. 131 –136.

[80] C. Sergiou and V. Vassiliou, "Performance evaluation of the DAlPaS congestion control algorithm in wireless sensor networks," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, june 2011, pp. 1–7.

[81] H. Hu, J. Min, X. Wang, and Y. Zhou, "The improvement of s-mac based on dynamic duty cycle in wireless sensor network," in *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, vol. 1, june 2011, pp. 341 –345.

[82] H. Yoo, M. Shim, and D. Kim, "Dynamic duty-cycle scheduling schemes for energy-harvesting wireless sensor networks," *Communications Letters, IEEE*, vol. 16, no. 2, pp. 202 –204, february 2012.

[83] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, " CODA: congestion detection and avoidance in sensor networks," in *Proc. 1st international conference on Embedded networked sensor systems* , 2003.

[84] C. J. Colbourn, M. Cui, E. L. Lloyd, and V. R. Syrotiuk, "A carrier sense multiple access protocol with power backoff (csma/pb)," *Ad Hoc Networks*, vol. 5, pp. 1233–1250, November 2007.

[85] H. Tao, R. Fengyuan, L. Chuang, and S. Das, "Alleviating congestion using traffic-aware dynamic routing in wireless sensor networks," in *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on*, june 2008, pp. 233 –241.

[86] Y. P. Hsu and K. T. Feng, "Cross-layer routing for congestion control in wireless sensor networks," in *Radio and Wireless Symposium, 2008 IEEE*, jan. 2008, pp. 783 –786.

[87] L. Y. Min and L. W. Yi, "Improved fairness using dccp in wireless sensor networks," in *Networking and Digital Society, 2009. ICNDS '09. International Conference on*, vol. 1, May 2009, pp. 141 –144.

[88] A. Dunkels, “The Contiki Operating System,” retrieved Jun 2012. [Online]. Available: http://www.sics.se/~adam/contiki/docs/main.html

[89] SenTools, “Contiki,” retrieved Jun 2012. [Online]. Available: http://senstools.gforge.inria.fr/doku.php?id=os:contiki

[90] A. Dunkels, “Rime: A lightweight layered communication stack for sensor networks,” in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, Jan. 2007. [Online]. Available: http://www.sics.se/~adam/dunkels07rime.pdf

[91] M. Durvy, J. Abeillé, P. Wetterwald, C. O’Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, “Making sensor networks ipv6 ready,” in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys ’08. New York, NY, USA: ACM, 2008, pp. 421–422. [Online]. Available: http://doi.acm.org/10.1145/1460412.1460483

[92] A. Dunkels, T. Voigt, J. Alonso, H. Ritter, and J. Schiller, “Connecting Wireless Sensornets with TCP/IP Networks,” in *Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC2004)*, Frankfurt (Oder), Germany, Feb. 2004, (C) Copyright 2004 Springer Verlag. http://www.springer.de/comp/lncs/index.html. [Online]. Available: http://www.sics.se/~adam/wwic2004.pdf

[93] A. Dunkels, T. Voigt, and J. Alonso, “Making TCP/IP Viable for Wireless Sensor Networks,” in *Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN 2004), work-in-progress session*, Berlin, Germany, Jan. 2004.

[94] A. Dunkels, “Full TCP/IP for 8 Bit Architectures,” in *Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)*, San Francisco, May 2003. [Online]. Available: http://www.sics.se/~adam/mobisys2003.pdf

[95] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, “Protothreads: simplifying event-driven programming of memory-constrained embedded systems,” in *Proceedings of the 4th international conference on Embedded networked sensor systems*, ser. SenSys ’06. New York, NY, USA: ACM, 2006, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182811

[96] Swedish Institute of Computer Science (SICS), "Why Contiki?" retrieved Jun 2012. [Online]. Available: http://www.sics.se/contiki/wiki/index.php/Why_Contiki%3F

[97] Wikipedia, "POSIX Threads," retrieved Jun 2012. [Online]. Available: http://en.wikipedia.org/wiki/POSIX_Threads

[98] IBM, "POSIX Threads Explained," retrieved Jun 2012. [Online]. Available: http://www.ibm.com/developerworks/library/l-posix1/index.html

[99] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level simulation in cooja," in *European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, 2007.

[100] F. Österlind, J. Eriksson, and A. Dunkels, "Cooja timeline: a power visualizer for sensor network simulation," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (ACM SenSys 2010)*, Zurich, Switzerland, 2010. [Online]. Available: http://www.sics.se/~adam/osterlind10cooja.pdf

[101] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón, "Towards interoperability testing for wireless sensor networks with cooja/mspsim," in *Proceedings of the 6th European Conference on Wireless Sensor Networks, EWSN 2009*, Cork, Ireland, Feb. 2009.

[102] Swedish Institute of Computer Science (SICS), "An Introduction to COOJA," retrieved Jun 2012. [Online]. Available: http://www.sics.se/contiki/wiki/index.php/An_Introduction_to_Cooja

[103] F. Österlind, "The COOJA Simulator," retrieved Jun 2012. [Online]. Available: http://www.sics.se/~fros/cooja.php

[104] F. Österlind, J. Eriksson, and A. Dunkels, "Cooja timeline: a power visualizer for sensor network simulation," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '10. New York, NY, USA: ACM, 2010, pp. 385–386.

[105] Sensinode, "Building the Embedded Web," retrieved Jun 2012. [Online]. Available: http://www.sensinode.com/EN/products.html

[106] "A True System on Chip solution for 2.4 GHz IEEE 802.15.4 / ZigBee®," CC2430 Data Sheet (rev. 2.1), May 2007.

[107] George Oikonomou, "8051 Memory Spaces," retrieved Jun 2012. [Online]. Available: http://www.sics.se/contiki/wiki/index.php/8051_Memory_Spaces

[108] IEEE802.15, "IEEE 802.15 WPAN Task Group 4 (TG4)," retrieved Jun 2012. [Online]. Available: http://www.ieee802.org/15/pub/TG4.html

[109] "Tmote sky: Ultra low power ieee 802.15.4 compliant wireless sensor module," Moteiv Corporation, Feb. 2006, tmote Sky Datasheet Revision 1.0.2.

[110] Institute for Telecommunication Sciences, "Telecommunications: Glossary of Telecommunication Terms," retrieved Jun 2012. [Online]. Available: http://www.its.bldrdoc.gov/fs-1037/dir-013/_1849.htm

[111] Wikipedia, "Duty Cycle," retrieved Jun 2012. [Online]. Available: http://en.wikipedia.org/wiki/Duty_cycle

[112] Famitracker, "Duty Cycle," retrieved Jun 2012. [Online]. Available: http://www.famitracker.com/wiki/index.php?title=Duty_cycle

[113] H. R. M. Warrick, E. Marder, A. I. Selverston, and M. Moulins, Eds., *Dynamic Biological Networks: The Stomatogastric Nervous System.* Cambridge, MA: MIT Press, 1992.

[114] Doctronics, "555 timer," retrieved Jun 2012. [Online]. Available: http://www.doctronics.co.uk/555.htm

[115] K. Rmer, P. Blum, and L. Meier, "Time synchronization and calibration in wireless sensor networks," 2005.

[116] M. Miklós, B. Kusy, S. Gyula, and L. Ákos, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 39–49.

[117] Swedish Institute of Computer Science (SICS), "Change MAC or Radio Duty Cycling Protocols," retrieved Jun 2012. [Online]. Available: http://www.sics.se/contiki/wiki/index.php/Change_MAC_or_Radio_Duty_Cycling_Protocols

[118] ——, "ContikiMAC," retrieved Jun 2012. [Online]. Available: http://www.sics.se/contiki/wiki/index.php/ContikiMAC

[119] IETF, "IPv6 over Low power WPAN (6lowpan)," retrieved Jun 2012. [Online]. Available: https://datatracker.ietf.org/wg/6lowpan/charter/

[120] Wikipedia, "6LoWPAN," retrieved Jun 2012. [Online]. Available: http://en.wikipedia.org/wiki/6LoWPAN

[121] J. W.Hui and D. E. Culler, "Extending ip to low-power, wireless personal area networks," *Internet Computing, IEEE*, vol. 12, no. 4, pp. 37–45, July-Aug. 2008.

[122] G. Mulligan, "The 6lowpan architecture," in *Proceedings of the 4th workshop on Embedded networked sensors*, ser. EmNets '07. New York, NY, USA: ACM, 2007, pp. 78–82.

[123] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet.* Wiley Publishing, 2010.

[124] S. Duquennoy, N. Wirstom, N. Tsiftes, and A. Dunkels, "Leveraging IP for Sensor Network Deployment," in *Proceedings of the workshop on Extending the Iternet to Low power and Lossy Networks (IP+SN 2011)*, Chicago, IL, USA, Apr. 2011. [Online]. Available: http://www.sics.se/~adam/duquennoy11leveraging.pdf

[125] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, "Making sensor networks ipv6 ready," in *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008), poster session*, Raleigh, North Carolina, USA, Nov. 2008, best poster award. [Online]. Available: http://www.sics.se/~adam/durvy08making.pdf

[126] A. Dunkels, "Full tcp/ip for 8-bit architectures," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, ser. MobiSys '03. New York, NY, USA: ACM, 2003, pp. 85–98.

[127] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, M. Durvy, J. P. Vasseur, A. Terzis, A. Dunkels, and D. Culler, "Beyond Interoperability: Pushing the Performance of Sensornet IP Stacks," in *Proceedings of the ACM Conference on Networked Embedded Sensor Systems, ACM SenSys 2011*, Seattle, WA, USA, Nov. 2011. [Online]. Available: http://www.sics.se/~adam/ko11beyond.pdf

[128] A. Dunkels, T. Voigt, J. Alonso, H. Ritter, and J. Schiller, "Connecting Wireless Sensornets with TCP/IP Networks," in *Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC2004)*, Frankfurt (Oder), Germany, Feb. 2004, (C) Copyright 2004

Springer Verlag. http://www.springer.de/comp/lncs/index.html. [Online]. Available: http://www.sics.se/~adam/wwic2004.pdf

[129] A. Dunkels, “Towards TCP/IP for Wireless Sensor Networks,” Licentiate thesis, Mar. 2005. [Online]. Available: http://www.sics.se/~adam/dunkels05towards.pdf

[130] *RPL: IPv6 Routing Protocol for Low power and Lossy Networks draft-ietf-roll-rpl-19*, IETF RFC, 2011.

[131] *An IPv6 Routing Header for Source Routes with RPL draft-ietf-6man-rpl-routing-header-07*, IETF RFC, 2011.

[132] T. W. (editor), P. T. (editor), B. Anders, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. P. Vasseur, and R. Alexander, “RPL: IPv6 Routing Protocol for Low power and Lossy Networks,” RFC 6550, Mar. 2012.

[133] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, A. Terzis, A. Dunkels, and D. Culler, “ContikiRPL and TinyRPL: Happy Together,” in *Proceedings of the workshop on Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, Chicago, IL, USA, Apr. 2011. [Online]. Available: http://www.sics.se/~adam/ko11contikirpl.pdf

[134] N. Tsiftes, J. Eriksson, and A. Dunkels, “Poster abstract: Low-power wireless ipv6 routing with contikirpl,” in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2010)*, Stockholm, Sweden, Apr. 2010. [Online]. Available: http://www.sics.se/~adam/tsiftes10rpl.pdf

[135] *The Trickle Algorithm draft-ietf-roll-trickle-08*, IETF RFC, 2011.

[136] T. Clausen, U. Herberg, and M. Philipp, “A critical evaluation of the ipv6 routing protocol for low power and lossy networks (rpl),” in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, oct. 2011, pp. 365 –372.

[137] L. Bergamini, A. V. Carlo Crociani, and M. Nati, “Validation of wsn simulators through a comparison with a real testbed,” in *Proceedings of the 7th ACM workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, ser. PE-WASUN ’10. New York, NY, USA: ACM, 2010, pp. 103–104. [Online]. Available: http://doi.acm.org/10.1145/1868589.1868611

[138] S. Hara, Z. Dapeng, K. Yanagihara, J. Taketsugu, K. Fukui, S. Fukunaga, and K. Kitayama, "Propagation characteristics of ieee 802.15.4 radio signal and their application for location estimation," in *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, vol. 1, may-1 june 2005, pp. 97 – 101 Vol. 1.

[139] Sourceforge, "The GNU Netcat," retrieved Jun 2012. [Online]. Available: http://netcat.sourceforge.net/

[140] OpenBSD, "Netcat manual pages," retrieved Jun 2012. [Online]. Available: http://www.openbsd.org/cgi-bin/man.cgi?query=nc

[141] J. Zhao, L. Wang, S. Li, X. Liu, Z. Yuan, and Z. Gao, "A survey of congestion control mechanisms in wireless sensor networks," *Intelligent Information Hiding and Multimedia Signal Processing, International Conference on*, vol. 0, pp. 719–722, 2010.

[142] R. Chakravarthi, C. Gomathy, S. K. Sebastian, P. K, and V. B. Mon, "A survey on congestion control in wireless sensor networks," *International Journal of Computer Science and Communication*, vol. 1, pp. 161–164, 2010.

[143] A. Dunkels, L. Mottola, N. Tsiftes, Fredrik Österlind, J. Eriksson, and N. Finne, "The Announcement Layer: Beacon Coordination for the Sensornet Stack," in *Proceedings of EWSN 2011*, Bonn, Germany, Feb. 2011.

[144] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," in *eprint arXiv:cs/9809099*, 2009.

[145] G. Oikonomou and I. Phillips, "Experiences from Porting the Contiki Operating System to a Popular Hardware Platform," in *Proc. 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, Barcelona, Spain, Jun. 2011.

[146] Lawrence Berkeley National Laboratory, "Data segment," retrieved Jun 2012. [Online]. Available: http://csg.lbl.gov/pipermail/vxwexplo/2004-January/004166.html

[147] Wikipedia, "Data segment," retrieved Jun 2012. [Online]. Available: http://en.wikipedia.org/wiki/Data_segment

[148] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 networks," RFC 4944, Sep. 2007.

[149] Hui (editor), Jonathan, and P. Thubert, “Compression format for IPv6 Datagrams over IEEE 802.15.4-Based Networks,” RFC 6282, Sep. 2011.

[150] N. Tsiftes, J. Eriksson, and A. Dunkels, “Low-power wireless ipv6 routing with contikirpl,” in *IPSN ’10: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks.* New York, NY, USA: ACM, 2010, pp. 406–407.

[151] J. Yiu and A. Frame, “8bit, 16bit or 32bit architecture is the better option?” ARM Holdings Plc, retrieved Jun 2012.

[152] “Sdcc - Small Device C Compiler,” retrieved 2011. [Online]. Available: http://sdcc.sourceforge.net/

[153] Sourceforge, “Sourceforge,” retrieved Jun 2012. [Online]. Available: http://sourceforge.net/

[154] Wikipedia, “DOT language,” retrieved Jun 2012. [Online]. Available: http://en.wikipedia.org/wiki/DOT_language

Appendix A

Data Collection at the Base Station

In this Chapter, the process and the format of data collected will be explained.

Sample of flow specific data collected by the UDP server:

```
<1331306022.646627> 2 bytes [00242]
from [2001:630:301:6453:215:2000:2:2147]:1027

<1331306022.678624> 2 bytes [00243]
from [2001:630:301:6453:215:2000:2:2147]:1027

<1331306022.722616> 2 bytes [00244]
from [2001:630:301:6453:215:2000:2:2147]:1027

<1331306022.770619> 2 bytes [00245]
from [2001:630:301:6453:215:2000:2:2147]:1027

<1331306022.818620> 2 bytes [00246]
from [2001:630:301:6453:215:2000:2:2147]:1027

<1331306022.874635> 2 bytes [00247]
from [2001:630:301:6453:215:2000:2:2147]:1027

<1331306022.914746> 2 bytes [00248]
from [2001:630:301:6453:215:2000:2:2147]:1027

<1331306022.958658> 2 bytes [00249]
from [2001:630:301:6453:215:2000:2:2147]:1027


2001:630:301:6453:215:2000:2:2147 74
2001:630:301:6453:215:2000:2:18ca 95

******** NEW CONGESTION EXECUTION ********
```

where the first field *(< 1331306022.646627 >)* is the time of reception, the second field *(2 bytes)* is the size of the data from the application, the third field *([00246])* is the contents of the data which is also a sequence number generated from the application at sensor nodes and the last two fields *([2001 : 630 : 301 : 6453 : 215 : 2000 : 2 : 2147] : 1027)* are the packet's source address and port number.

Moreover, when requested packet flows and thus the experiment ends; two lines showing the number of packets received from each source is printed followed by a line signaling the beginning of a new experiment run. This way, multiple experiments can be saved in the same file without trouble to distinguish the results between them.

Sample of periodic data collected by the UDP server:

```
<1330515978.762626> 20 bytes (seq:cpuh:irq:tx:rx)
| 107:5171189:1181510:832069:639214
| from [2001:630:301:6453:215:2000:2:2159]:1028

<1330515979.258602> 20 bytes (seq:cpuh:irq:tx:rx)
| 107:2864707:1155133:255345:222854
| from [2001:630:301:6453:215:2000:2:18ca]:1028

<1330515980.766605> 20 bytes (seq:cpuh:irq:tx:rx)
| 107:2843220:1178512:328676:555923
| from [2001:630:301:6453:215:2000:2:210e]:1028

<1330516014.638611> 20 bytes (seq:cpuh:irq:tx:rx)
| 108:4193694:1243937:743822:733225
| from [2001:630:301:6453:215:2000:2:19b1]:1028

<1330516019.78625> 20 bytes (seq:cpuh:irq:tx:rx)
| 108:3125058:1207536:305312:798091
| from [2001:630:301:6453:215:2000:2:1809]:1028

<1330516019.462607> 20 bytes (seq:cpuh:irq:tx:rx)
| 108:2538966:1199609:28949:598573
| from [2001:630:301:6453:215:2000:2:18b9]:1028

<1330516020.606617> 20 bytes (seq:cpuh:irq:tx:rx)
| 108:4108486:1188302:198611:566071
| from [2001:630:301:6453:215:2000:2:1947]:1028

<1330516027.346610> 20 bytes (seq:cpuh:irq:tx:rx)
| 108:3358688:1186318:591719:632757
| from [2001:630:301:6453:215:2000:2:1a29]:1028

<1330516029.542607> 20 bytes (seq:cpuh:irq:tx:rx)
| 108:4294329:1242582:450511:617341
| from [2001:630:301:6453:215:2000:2:2147]:1028

<1330516031.86611> 20 bytes (seq:cpuh:irq:tx:rx)
| 108:2719852:1182352:193885:508307
| from [2001:630:301:6453:215:200:2:18b1]:1028
```

Where the first field *(< 1330515978.762626 >)* is the time of reception, the second field *(20 bytes)* is the size of the data from the application, the third field *(107 5171189 1181510 832069 639214)* is the contents of the data. In this type of messages, the data received are the time spend at each of the CPU-hi, IRQ, radio RX and radio TX states of the sensor nodes. Finally, *([2001 : 630 : 301 : 6453 : 215 : 2000 : 2 : 2147] : 1027)* are the packet's source address and port number.

In general, it is very hard to synchronise the clocks in different devices such as sensor nodes. Consequently, during the test-bed experiments, ping6 and thus round-trip delay was measured. In order to measure round trip delay with different traffic loads the command:

```
 ping6 -i <packet transmission interval> <IPv6 address>
```

was used. The argument -i specifies the interval between the consequent packet transmissions. An example of ping6 used for the measurement of round trip delay in our test-bed is presented:

```
PING 2001:630:301:6453:215:2000:2:19b1 56 data bytes

64 bytes from 2001:630:301:6453:215:2000:2:19b1:
icmp\_seq=1 ttl=64 time=25.2 ms

64 bytes from 2001:630:301:6453:215:2000:2:19b1:
icmp\_seq=2 ttl=64 time=23.9 ms

64 bytes from 2001:630:301:6453:215:2000:2:19b1:
icmp\_seq=3 ttl=64 time=21.9 ms
64 bytes from 2001:630:301:6453:215:2000:2:19b1:
icmp\_seq=4 ttl=64 time=24.0 ms

64 bytes from 2001:630:301:6453:215:2000:2:19b1:
icmp\_seq=5 ttl=64 time=22.9 ms

64 bytes from 2001:630:301:6453:215:2000:2:19b1:
icmp\_seq=6 ttl=64 time=25.9 ms

64 bytes from 2001:630:301:6453:215:2000:2:19b1:
icmp\_seq=7 ttl=64 time=23.9 ms

--- 2001:630:301:6453:215:2000:2:19b1 ping statistics ---
7 packets transmitted, 7 received, 0\% packet loss, time 6009ms
rtt min/avg/max/mdev = 21.928/23.985/25.929/1.244 ms
```

After the collection of the desired data, a Linux shell script was used for the extraction and the formating of the data in one file. This file was the inserted to Microsoft Excel and processed with the use of pivot tables.

Appendix B

Test-bed Tools

Some tools, essential for the debugging and information gathering from the test-bed will be described in this Chapter. These tools are:

- **Bootloader, booty:** nano-usb-programmer *(the software used for loading the firmware to sensor devices)* cannot work on OS X and will only work in slow mode on Linux. Operating in slow mode in Linux can take a significant amount of time in order to programm sensor devices and thus it is not practical for loading firmware to a large number of nodes.

  In order to program from linux or OS X, bootty and bootloader can be used. For these programmes a working SDCC compiler is required on the PC, with libraries built for model-small. SDCC build instructions can provide more information about the SDCC models [152].

  The bootloader has two parts:

  bootty is the embedded part. This part is loaded to the sensors. ball is the host part. This is what you run on your PC.

  After booty is loaded to the sensors *(with nano-usb-programmer)*, ball can be used from Linux to load any firmware to the sensor device. This can be done with the command:

  ```
  ./ball -s -W -f <contiki-image.ihx> <device name>
  ```

  Where device name will be the Port in which the device is connected to.

  How this works in detail follows: When a sensor is carrying bootty, as soon as it's turned on it will start with the bootloader.

If you wait for 10 seconds, the bootloader will jump to the application *(contiki in our case)*. Pressing Button 1 will jump to the application directly - no need to wait. Pressing Button 2 will suspend the timer. Led 2 will toggle when pressed, to confirm that the timer got suspended. Pressing it again will NOT resume the timer. Suspending the timer gives the time to do processes such as read mac and upload external images. Press Button 1 will result into jumping to the application sitting in the internal flash.

What are the advantages and disadvantages of using booty and bootloader.
Disadvantages:

1. Having bootloader at sensors consumes some memory. Therefore, less space for the applications is left at the sensors.

Advantages:

1. Nodes can now be programmed with ease from any operating system.
2. Multiple images can now be loaded at the external flash of the sensor *(Sensinode devices have 2MB of external flash. This means that over 15 images can be loaded to each sensor device)* and switching between different images can now be done by rebooting the devices.
3. Combining booty with an auto sensor programming script, can result in flashing multiple devices at the same time.

Booty and bootloader can be found as an open source project at [153].

- **Auto sensor programming script:** If the sensor devices have been loaded with bootloader, this script can be used for the configuration of the devices and the automation of the image loading at sensors and developed in Loughborough University.

  The script used for the automated flashing of multiple devices is:

```bash
#!/bin/bash

# $1 must be the full path to the
#file to be loaded on sensinode minus (-number.ihx)
#for example if the files are testbed-0.ihx
#... testbed-4.ihx $1 must end in testbed
# $2 must be the total number of images
#to be uploaded in the node

for file in /dev/ttyUSB*
```

```
do

  ./ball -m -s $file
  if [ $? -ne 0 ]
  then
    echo "read mac failed"
  exit 1
  fi

  ./ball -T 0 -s $file
  if [ $? -ne 0 ]
  then
    echo "delete register failed"
  exit 1
  fi

  ./ball -F -s $file
  if [ $? -ne 0 ]
  then
    echo "format failed"
  exit 1
  fi

  ./ball -W -f $1-0.ihx -s $file
  if [ $? -ne 0 ]
  then
    echo "error uploading image"
  exit 1
  fi

  for (( i = 0; i < $2; i++ ))
  do
      ./ball -w $i -f $1-$i.ihx -s $file
      if [ $? -ne 0 ]
      then
        echo "error writing image : $1 to external flash"
        exit 1
      fi
  done
done
```

This script first reads the mac address of the connected through the serial port device. If there is no mac address the script terminates. Then the script attempts to write the to the status register. If writing to the status register fails the script terminates. Next the scripts attempt to erase the contents of the flash memory. Similarly to the previous checks the script will Terminate in case of error. Finally, the script will attempt to write the input file to the flash and multiple other images *(taken as an input)* to the external flash. If any of the writing procedures fail, the script will terminate and return an error.

- **Energy scan:** Energy scan is a built in tool of Contiki OS *(can be found in /examples/sensinode/ directory)*. When uploaded in a sensor node, the node will sample all of the 802.15.4 channels *(11 - 26)* and return the highest RSSI value measured during the channel sampling process.

A node configured with energy scan will print in the terminal an output as the one sheen below:

```
===============
11 [-49]: ##
12 [-49]: ##
13 [-49]: ##
14 [-23]: ############################
15 [-49]: ##
16 [-49]: ##
17 [-36]: ###############
18 [-48]: ###
19 [-42]: #########
20 [-49]: ##
21 [-48]: ###
22 [-48]: ###
23 [-49]: ##
24 [-49]: ##
25 [-48]: ###
26 [-49]: ##
===============
11 [-49]: ##
12 [-49]: ##
13 [-49]: ##
14 [-49]: ##
15 [-49]: ##
16 [-49]: ##
17 [-35]: ################
18 [ -7]: ############################################
19 [-29]: ######################
20 [-49]: ##
21 [-49]: ##
22 [-48]: ###
23 [-49]: ##
24 [-49]: ##
25 [-49]: ##
26 [-49]: ##
===============
11 [-48]: ###
12 [-15]: ####################################
13 [-49]: ##
14 [-49]: ##
15 [-49]: ##
16 [-44]: #######
17 [-49]: ##
18 [-48]: ###
19 [-48]: ###
20 [-49]: ##
21 [-48]: ###
22 [-48]: ###
23 [-49]: ##
24 [-49]: ##
25 [-48]: ###
26 [-49]: ##
===============
```

Where the first number represents the channel, the negative number within squared brackets represents the RSSI value measured and the hashes represent how busy each channel was.

- **Packet sniffer and Wireshark:** Sniffer is Contiki OS built-in tool for

CC24XX ports. It's purpose is to capture all the traffic in the medium.

A sniffer node captures all the traffic from the radio medium *(sniffer is configured to capture traffic from a specific 802.15.4 channel and thus cannot capture 802.15.4 traffic from every channel)* and pipe it down the serial line. Sniffer nodes does not participate in the network and can be connected to any PC.

The functionality is controlled by the 'CC2430_RF_CONF_HEXDUMP' define. This define can be configured from contiki-conf.h or from the local project-conf.h of the sniffer directory.

When you set this define to 1, the radio driver will output hexdumps for all traffic.

In order to perform a live network analysis, the traffic must be directed to Wireshark in real-time. Text2pcap *(converts a plain text file to pcap format)* cannot do this conversion in real-time. In the official Contiki OS for CC243XX devices another tool with the name of n601-cap is included to perform this task. This tool can be downloaded at [153] by searching for tools in Contiki OS for CC24XX platforms. N601-cap reads a hexdump and pipes it to Wireshark. The hexdump can come from stdin or it can come directly from a serial line. Communication with Wireshark happens via a FIFO pipe [107].

How n601-cap works:

First n601-cap should be downloaded. Then user must enter the directory with the command cd n601-cap *(assuming it is downloaded to the directory the user is currently at)*. The command 'make' should then be invoked in order to make the executables. Then the executable should be used as shown below:

```
./n601-cap -d /dev/ttyUSBx
```

Where the -d argument indicates that the device is connected to /dev/ttyUSBx interface. X in USBx should be replaced with the actual USB number where your device is connected.

If permission for execution is denied, the above command should be used with Sudo *(administrator rights)*.

All the procedures are now set and traffic can be directed in Wireshark by: go to Capture $->$ Options and manually type /tmp/n601 in the interface box. Alternatively, Wireshark can be invoked from the command line by typing:

```
wireshark -k -i /tmp/n601
```

- **Viztool:** this tool have been developed in Loughborough University and can be found at http://nets/gitweb/ *(for users that already have permissions)*. The purpose of this tool is to collect routing information from nodes in a RPL and 6LoWPAN network and demonstrate this information in an easy to read format.

  Someone has to first download viztool from the previously mentioned directory and make the project. After making the project, viztool is ready to be used. The command:

  ```
  ./viztool <IPv6 address>
  ```

  will request all the information saved in the routing table of the node with the <IPv6 address> address used as the viztool input argument. Additionally, if the requested node *(viztool input address)* has any route entries in it's routing table; viztool will request the same information from these entries. Therefore using the IPv6 address of the RPL route node as an input argument for viztool, will result in retrieving the above mentioned information from every node in the network *(assuming there is only one RPL DODAG)*.

  When viztool receives the requested information, it generates a Dot [154] file. Using the command:

  ```
  dot -Tpdf -O <name of the dot file generated by viztool>
  ```

  dot will automatically generate two pdf files. One of them, represents the routing tree while the other represents a link layer map *(which nodes are in transmission range of each other)*.

Appendix C

Simulation Automations and Result Mining

In this Chapter, detailed information about the automation of experiments in COOJA simulator will be presented.

By default, COOJA can perform simulations ether through a graphical interface or the terminal. Performing network simulations through the graphical interface; allows user to monitor the networks behaviour during simulation time. Even though this is very convenient for debugging purposes, it can take significantly longer than simulating without graphical interface. Additionally, COOJA provides a built in automation script *(only for executions through terminal)* which is capable of repeating the same experiment multiple times. This script can be found in:

```
/contikiXXX/tools/cooja/contiki_tests
```

directory. Running COOJA simulations repeatedly can be achieved using the command:

```
bash RUN_REPEATED <Number of repeats> <COOJA simulator file>
```

Using the RUN_REPEATED script of COOJA can significantly reduce the amount of time required to perform a gret number of simulations. The draw back of this approach is that it does not allow any modifications in the Contiki codes executed by COOJA during the simulation repeats. To achieve that and thus totally automate COOJA *(reconfigure the simulated nodes with different parameters between the simulation iterations)*, a number of modifications -additions are required. This required changes are presented below:

1. First the RUN_REPEATED script of COOJA will have to be modified in order to export in the Linix a variable representing which simulation irritation COOJA is currently in. Moreover, the script should be modified to recompile all the files in Contiki before each simulation repeat *(traditionally*

if there are no files changed, COOJA will not recompile Contiki). The modified to achieve the above mentioned characteristics script of COOJA can be seen below:

```
#!/bin/bash

AUTO_CONF=cooja-conf.h

# Usage
if [ $# -eq 2 ]; then
  REPEATS=$1
  TEST=$2
else
  echo "Usage: $0 <nr_repeats> <test>"
  echo "Example: $0 10 cooja_helloworld"
  exit 1
fi

# Locate Contiki/COOJA
if [ -z "$CONTIKI" ]; then
  if [ -z "$CONTIKI_HOME" ]; then
        CONTIKI_HOME=../../..
  fi
  CONTIKI=$CONTIKI_HOME
fi

# Clean up
rm -f *.log *.cooja_log
rm -fr se obj_cooja
rm -f symbols.c symbols.h

# Compile COOJA
echo ">>>>>>> Building COOJA <<<<<<<<"
(cd $CONTIKI/tools/cooja && ant clean && ant jar)
if [ "$?" != "0" ]; then
  echo "Compilation of COOJA failed"
  exit 1
fi

# Run tests
export BOX_NUM=`expr substr \`hostname\` 2 1`
SRC_DIR=$(dirname "$TEST")
TEST_NAME=$(basename "$TEST")
mkdir -p $TEST-$BOX_NUM
for COUNTER in `seq 1 $REPEATS`;
do
  echo ">>>>>>> Test $COUNTER/$REPEATS: \
   $TEST-$COUNTER.log <<<<<<<<"
  export COUNTER=$COUNTER
  echo ">>>>>>> touch -c $SRC_DIR/$AUTO_CONF <<<<<<<<"
  touch -c $SRC_DIR/$AUTO_CONF
  bash RUN_TEST $TEST RUN_REPEATED_LAST.log
  mv $TEST.log $TEST-$BOX_NUM-$COUNTER.log
  mv $TEST-$BOX_NUM-$COUNTER.log $TEST-$BOX_NUM
done

echo
cat RUN_REPEATED_LAST.log
cd $SRC_DIR
tar zcf $TEST_NAME-$BOX_NUM.tar.gz $TEST_NAME-$BOX_NUM
cd -
echo
echo ">>>>>>> DONE! Test logs stored \
in $TEST-[1-$REPEATS].log <<<<<<<<"
```

2. The projects makefile has to be modified to pass the previously exported by COOJA *(and the virtual PC's that the simulations will be executed at. In our case this is handled by "BOX_NUM")*, number of experiment repeat in Contiki. This can be achieved by adding the in the beginning of the projects makefile:

```
ifdef COOJA
  DEFINES+=PROJECT_CONF_H=\"project-conf.h\",COOJA=$(COOJA)
  ifdef COUNTER
    DEFINES+=COUNTER=$(COUNTER)
  endif
  ifdef BOX_NUM
    DEFINES+=BOX_NUM=$(BOX_NUM)
  endif
else
  DEFINES+=PROJECT_CONF_H=\"project-conf.h\"
endif
```

A cooja-conf.h file has to be created. This file will be responsible of reconfiguring the nodes between each COOJA run. The reconfiguration is based in the previously exported by cooja number of simulation repeat. An example of a cooja-conf.h file can be seen below *(and the one used during our simulations)*:

```
/*activate configuration for automated experiments*/
#define EXPERIMENT 1

#define PACKETS_TO_SEND 500 /* Stop seeding after 500 packets */


/* What algorithm will each box run */
#if BOX_NUM == 1
#undef NETSTACK_CONF_MAC
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_MAC CADC_driver
#define NETSTACK_CONF_RDC CADCrdc_driver
#define MAC_USED 0
#elif BOX_NUM == 2
#undef NETSTACK_CONF_MAC
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_MAC beam_driver
#define NETSTACK_CONF_RDC beamrdc_driver
#define MAC_USED 0
#elif BOX_NUM == 3
#undef NETSTACK_CONF_MAC
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_MAC csma_driver
#define NETSTACK_CONF_RDC contikimac_driver
#define MAC_USED 1
#elif BOX_NUM == 4
#undef NETSTACK_CONF_MAC
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_MAC csma_driver
#define NETSTACK_CONF_RDC cxmac_driver
#define MAC_USED 1
#else
#undef NETSTACK_CONF_MAC
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_MAC csma_driver
```

```
#define NETSTACK_CONF_RDC lpp_driver
#define MAC_USED 1
#endif

#define DIVISOR 12
/* Convert the cooja script's
$COUNTER to something that makes sense */
#ifdef COUNTER
#define CONF ((COUNTER - 1) % DIVISOR)
#else
#define CONF 0
#endif

/* What delay will each box use */
#if (CONF % 4 == 0)
#define SIM_CONF_SEND_INTERVAL 64
#define SEND_INTERVAL   64
#elif (CONF % 4 == 1)
#define SIM_CONF_SEND_INTERVAL  32
#define SEND_INTERVAL   32
#elif (CONF % 4 == 2)
#define SIM_CONF_SEND_INTERVAL  16
#define SEND_INTERVAL   16
#elif (CONF % 4 == 3)
#define SIM_CONF_SEND_INTERVAL  8
#define SEND_INTERVAL   8
#endif

/*number 1 is the server we dont need it*/
#define NEW_SOURCE (((COUNTER-1) % 9) + 2)

#if MAC_USED
/* Pairwise configurations: 5 per RDC */
#define PAIR ((COUNTER-1) % 5)

#if PAIR==0
#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 4
#elif PAIR==1
#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 8
#elif PAIR==2
#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 16
#elif PAIR==3
#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 32
#else
#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 64
#endif
#else
#define PAIR ((COUNTER-1) % 2/*3*/)
#if PAIR==0
#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 4
/*#elif PAIR==1*/
#else
#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 8
/*
#else
#undef NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 16
```

```
*/
#endif
#endif
```

This file will change the MAC and RDC layer as well as the inter packet transmission interval and the channel check rate used by the protocols in each simulation run.

3. Some codes in Java script must be written and imported in COOJA's Contiki test editor module in order to automatically collect and save *(preferable in an easy to manage format)* all of the required results from each experiment. A Java script example used during our simulation experiments can be seen below:

```
TIMEOUT(600000, log.testOK());

function Network() {
  var n = new Array(); /* Nodes */
  var p = new Array(); /* Packets */


  this.add = function(node) {
    n.push(node);
  }

  this.newPacket = function(packet) {
    p.push(packet);
  }

  this.getPacketSentTime = function(id) {
    var foo = parseInt(id);
    for(i = 0; i < p.length; i++) {
      if(p[i].getID() == foo) {
        return p[i].getSent();
      }
    }
  }

  this.report = function() {
    n.sort(sortID);
    for(i = 0; i < n.length; i++) {
      n[i].print();
    }
  }

  this.getByID = function(id) {
    for(i = 0; i < n.length; i++) {
      if(n[i].getID() == id) {
        return n[i];
      }
    }
  }


  function sortID(a, b) {
    return a.getID() - b.getID();
  }
}

function Node(id, t) {
  var id = id;
  /* var type;*/
  var packet_delay_tot = 0;


   this.rx = 0;
  this.tx = 0;
  //this.degree = 0;
  this.eCPU = 0;
  this.eIRQ = 0;
  this.eLPM = 0;
  this.eRX = 0;
  this.eTX = 0;
  this.highest = 0;
  this.mac = 0;
  this.interval = 0;
  this.channelcheck = 0;
  this.experiment = 0;
  this.rdc = 0;



 /* this.getType = function() {
    return type;
  }*/

  this.getID = function() {
    return id;
  }

  this.delay_incr = function(t) {
    packet_delay_tot += t;
  }

  this.print = function() {
    log.log(this.experiment + ";" + id + ";"
     + this.mac + ";" + this.rdc + ";"   +
      this.channelcheck + ";" + this.interval + ";");
    log.log(this.rx + ";" + this.tx + ";");
    log.log(this.eCPU + ";" + this.eIRQ + ";"
     + this.eLPM + ";" + this.eRX + ";" + this.eTX + ";");
    log.log(this.getAvgPacketDelay() + "\n");
  }

  this.updateHopCount = function(ttl) {
    hops = 64 - ttl + 1;
    if(hops < hc) { hc = hops; }
  }

  this.getAvgPacketDelay = function() {
    if(this.rx == 0) {
      return 0;
    } else {
      return (packet_delay_tot/this.rx/1000000);
    }
  }
}

function Packet(i, s) {
  var id = parseInt(i);
  var sent = s;

  this.getID = function() {
    return id;
  }

  this.getSent = function() {
    return sent;
  }
}

net = new Network();
sim = mote.getSimulation();

for(i = 0; i < sim.getMotesCount(); i++) {
  us = sim.getMote(i);
  n = new Node(us.getID() , us.getType().getDescription());
  net.add(n);
}

while (1) {
  /* Yield-ish */
  SEMAPHORE_SIM.release();
  SEMAPHORE_SCRIPT.acquire();
  if (SHUTDOWN) { net.report(); SCRIPT_KILL(); }
  if (TIMEOUT) { net.report(); SCRIPT_TIMEOUT(); }
  msg = new java.lang.String(msg);
  node.setMoteMsg(mote, msg);

  n = net.getByID(mote.getID());
  msgArray = msg.split(' ');
  if(msgArray[0].equals("DATA1")) {
    p = new Packet(msgArray[1], time);
    n.tx++;
    net.newPacket(p);
  } else if(msgArray[0].equals("DATA2")) {
    n.rx++;
       /* Packet Arrival Delay */
    t_diff = time - net.getPacketSentTime(msgArray[2]);
    n.delay_incr(t_diff);

  } else if(msgArray[0].equals("Periodic:")) {
    n.eCPU = msgArray[1];
    n.eIRQ = msgArray[2];
    n.eLPM = msgArray[3];
    n.eRX = msgArray[4];
    n.eTX = msgArray[5];
  } else if(msgArray[0].equals("Conf:")) {
    /* Algorithm - Counter - Messages
     - Period - RDC - Energy */
    n.mac = msgArray[1];
    n.channelcheck = msgArray[2];
    n.interval = msgArray[3];
    n.experiment = msgArray[4];
    n.rdc = msgArray[5];
  }
}
```

It is worthy to mention that the above Java script reads the first word printed by each node's printf function and based on that saves the desired information for each node. When the simulation ends, the Java script will print and save in a log file all the information gathered during the simulation run. The output will be one line for each simulated node.

Appendix D

Sensinode - CC2430 Constrains and Code Optimisations

During section 6.7 and section 3.10 lots of the hardware specific constrains were discussed. This Chapter will describe in detail the majority of code optimisations required in order to have a fully functional, IPv6 and 6LoWPAN ready; Contiki for Sensinode devices.

First why 15 nodes were used during the test-bed experiments will be explained while a general discussion describing why architectures based on 8051 MCU need this stack optimisations will follow. Since this research study's focus is not Contiki's code optimisation, only the related to duty cycle and this research optimisations will be presented afterwards. The remainder of Contiki's optimisations for 8051 MCU based architectures and thus Sensinode can be found in [145].

Information such as static variables are saved in RAM in 8051 MCU. Therefore all routing tables will be saved in the SRAM in our sensor nodes. CC24XX which is used by our sensor nodes, is based on 8051 MCU and has 8KB of RAM. During our experiments, when we increased the size of the routing and neighbor table to contain more than 15 entries, the size of the information could not fit in the 8KB of SRAM and thus the hardware could not function. This lead us into conducting test-bed experiments with up to 15 nodes.

Section 3.4 explains in detail the 8051 MCU and it's memory spaces. The fast access read/write data memory *256 Bytes* in 8051 MCU used by CC24XX SOC; is used as the machine stack. The main purpose of the machine stack is to keep track of the point to which each active subroutine should return control when it finishes executing. An active subroutine is one that has been called but is yet to complete execution after which control should be handed back to the point of call. Such activations of subroutines may be nested to any level (recursive as a special case), hence the stack. The most important *(related to our project)* of the program's information that need to be stored in the stack follows:

- **The return address.** When a subroutine is called, the location *(address)* of the instruction at which it can later resume the execution is saved in the stack.

- **Local data storage.** A subroutine frequently needs memory space for storing the values of local variables, the variables that are known only within the active subroutine and do not retain values after it returns.

- **Parameter passing.** Subroutines often require that values for parameters be supplied to them by the code which calls them, and it is not uncommon that space for these parameters may be laid out in the call stack.

- **Evaluation stack.** Operands for arithmetic or logical operations are most often placed into registers and operated on the stack.

Taking the above under consideration it is easy to say that the more function calls the more space required in the stack. If the space of the information that need to be stored in the stack exceed the stacks capacity; in most cases the hardware will stop working or even crash.

8051 based MCUs have only 256 byte of stack. If a program requires many nested function calls *(When Contiki OS is configured with IPv6 this is the case)*, 256 bytes of stack may not be enough. Therefore, stack optimisations in the codes must be performed in order to avoid potential crashes at the nodes. The stack optimisations related to this project follows:

- A great number of local variables within multiple functions in Contiki have been redefined as static. This forces this variables to be saved in the external RAM of 8051 MCU and thus we avoid saving them in the stack. It is worthy to mention, that for multiple architectures such as the ones based on MSP430 this approach would have the opposite effect. Architectures such as the ones based on MSP430 use the non used by the program RAM as their stack. Therefore saving variables on the RAM will result in a smaller stack.

- Modified the clock interrupt. By changing the value of CLOCK_CONF_ACCURATE to 0 *(can be found in platform/sensinode/ contiki-conf.h)* the the clock interrupt will now directly call the related functions. Instead a flag will be set and the required functions will be called by main. The related to this modifications codes follows:

```
/************ In contiki-sensinode-main.c file *******/
.
.
```

```
.
/************ Various unrelated codes *******/
.
.
.
while(1) {
    do {
      /* Reset watchdog and handle polls and events */
      watchdog_periodic();

      /**/
#if !CLOCK_CONF_ACCURATE
      if(sleep_flag) {
        if(etimer_pending() &&
            (etimer_next_expiration_time()
             - count - 1) > MAX_TICKS)
            { /*core/sys/etimer.c*/
          etimer_request_poll();
        }
        sleep_flag = 0;
      }
#endif.
.
.
.
/************ Various unrelated codes *******/
.
.
.
/************ In clock.c file *******/
#include <stdio.h> /*for debug printf*/
#include "sys/clock.h"
#include "sys/etimer.h"
#include "cc2430_sfr.h"
#include "sys/energest.h"

/*Sleep timer runs on the 32k RC osc. */
/* One clock tick is 7.8 ms */
#define TICK_VAL (32768/128)  /* 256 */

/* Used in sleep timer interrupt for
 calculating the next interrupt time */

static unsigned long timer_value;
/*starts calculating the ticks right after reset*/
#if CLOCK_CONF_ACCURATE
static volatile __data clock_time_t count = 0;
#else
volatile __data clock_time_t count = 0;
/* accurate clock is stack hungry */
volatile __bit sleep_flag;
#endif
/*calculates seconds*/
static volatile __data clock_time_t seconds = 0;
.
.
.
/************ Various unrelated codes *******/
.
.
.
void clock_ISR( void ) __interrupt (ST_VECTOR)
{
  IEN0_EA = 0;  /*interrupt disable*/
  ENERGEST_ON(ENERGEST_TYPE_IRQ);

  /*
   * If the Sleep timer throws an interrupt
    while we are powering down to
   * PM1, we need to abort the power down.
    Clear SLEEP.MODE, this will signal
   * main() to abort the PM1 transition
   */
  SLEEP &= 0xFC;

  /* When using the cooperative
     scheduler the timer 2 ISR is only
     required to increment the RTOS
     tick count. */

  /*Read value of the ST0,ST1,ST2 and
   then add TICK_VAL and write it back.
    Next interrupt occurs after the
     current time + TICK_VAL*/
  timer_value = ST0;
  timer_value += ((unsigned long int)ST1) << 8;
  timer_value += ((unsigned long int)ST2) << 16;
  timer_value += TICK_VAL;
  ST2 = (unsigned char) (timer_value >> 16);
  ST1 = (unsigned char) (timer_value >> 8);
  ST0 = (unsigned char) timer_value;

  ++count;

  /* Make sure the CLOCK_CONF_SECOND is a power of two,
         to ensure that the modulo operation below becomes
         a logical and not an expensive divide.
         Algorithm from Wikipedia:
     http://en.wikipedia.org/wiki/Power_of_two */
#if (CLOCK_CONF_SECOND & (CLOCK_CONF_SECOND - 1)) != 0
#error CLOCK_CONF_SECOND must be a power of two
#(i.e., 1, 2, 4, 8, 16, 32, 64, ...).
#error Change CLOCK_CONF_SECOND in contiki-conf.h.
#endif
  if(count % CLOCK_CONF_SECOND == 0) {
    ++seconds;
  }

#if CLOCK_CONF_ACCURATE
  if(etimer_pending() &&
     (etimer_next_expiration_time() - count - 1) > MAX_TICKS)
     {  /*core/sys/etimer.c*/
    etimer_request_poll();
  }
#else
  sleep_flag = 1;
#endif

  IRCON &= ~STIF;
  /*IRCON.STIF=Sleep timer interrupt flag.
  * This flag called this interrupt
  * func, now reset it*/
  ENERGEST_OFF(ENERGEST_TYPE_IRQ);
  IEN0_EA = 1;          /*interrupt enable*/
}
```

- Modified the r-timer interrupt. The r-timer interrupt is used by ContikiMAC *(and some other duty cycling schemes)* in order to accurately wake up the radio. If the stack is high there is a high possibility for stack overflow when the related to the interrupt function calls invoked. Therefore, the interrupt function have been modified to first perform a stack check. If there is a risk of stack overflow, the execution of the related to the interrupt function calls is rescheduled *(for microseconds)*. The related to this modifications codes follows:

```
/************ rtimer-arch.c file *******/
.
.
.
/************ Various unrelated codes *******/
.
.
.
void cc2430_timer_1_ISR(void) __interrupt (T1_VECTOR)
{
  IEN1_T1IE = 0; /* Ignore Timer 1 Interrupts */
  ENERGEST_ON(ENERGEST_TYPE_IRQ);
  /* No more interrupts from Channel 1 till next
   * rtimer_arch_schedule() call.
   * Setting the mask will instantly
   * generate an interrupt so we clear the
   * flag first. */
  T1CTL &= ~(CH1IF);
  T1CCTL1 &= ~T1IM;

  if(SP<0xa0) {
    rtimer_run_next();
  } else {
    rtimer_arch_schedule(0);
  }

  ENERGEST_OFF(ENERGEST_TYPE_IRQ);
  IEN1_T1IE = 1; /* Acknowledge Timer 1 Interrupts */
}
```

Appendix E

CADC 802.15.4 Packet Framer Modifications

Subsection 6.3.3 described in detail that for the propagation of channel check rate information, CADC uses the 3 reserved bits of the FCF field in the 802.15.4 frame header. Before each unicast transmission, nodes will set a flag in the 3 bit field. Through that flag, a node can calculate the channel check rate at its child nodes. In order to succeed doing this, we first created a new attribute related to congestions indication in Contikis packet buffer:

```
enum {
/************ packetbuf.c file *******/
.
.
.
/************ Various unrelated codes *******/
.
.
.
  PACKETBUF_ATTR_NONE,

  /* Scope 0 attributes: used only on the local node. */
  PACKETBUF_ATTR_CHANNEL,
  PACKETBUF_ATTR_NETWORK_ID,
  PACKETBUF_ATTR_LINK_QUALITY,
  PACKETBUF_ATTR_RSSI,
  PACKETBUF_ATTR_TIMESTAMP,
  PACKETBUF_ATTR_RADIO_TXPOWER,
  PACKETBUF_ATTR_LISTEN_TIME,
  PACKETBUF_ATTR_TRANSMIT_TIME,
  PACKETBUF_ATTR_MAX_MAC_TRANSMISSIONS,
  PACKETBUF_ATTR_MAC_SEQNO,
  PACKETBUF_ATTR_MAC_ACK,

  /* Scope 1 attributes: used between two neighbours only. */
  PACKETBUF_ATTR_RELIABLE,
  PACKETBUF_ATTR_PACKET_ID,
  PACKETBUF_ATTR_PACKET_TYPE,
  PACKETBUF_ATTR_REXMIT,
  PACKETBUF_ATTR_MAX_REXMIT,
  PACKETBUF_ATTR_NUM_REXMIT,
  PACKETBUF_ATTR_PENDING,
  /*****Congestion attribute***********/
  PACKETBUF_ATTR_CONGESTION,

  /* Scope 2 attributes: used between end-to-end nodes. */
  PACKETBUF_ATTR_HOPS,
  PACKETBUF_ATTR_TTL,
  PACKETBUF_ATTR_EPACKET_ID,
  PACKETBUF_ATTR_EPACKET_TYPE,
  PACKETBUF_ATTR_ERELIABLE,

  /* These must be last */
  PACKETBUF_ADDR_SENDER,
  PACKETBUF_ADDR_RECEIVER,
  PACKETBUF_ADDR_ESENDER,
  PACKETBUF_ADDR_ERECEIVER,

  PACKETBUF_ATTR_MAX
};
```

The congestion attribute of Contiki's packet buffer is updated by the MAC layer before a packet transmission and RDC when a packet is received.

The files related with 802.15.4 framer had to be modified as well in order to successfully save in the packets header the information carried in of PACKETBUF_ATTR_CONGESTION. The codes related to the modification of the 802.15.4 framer are:

```
/************ frame802154.c file *******/
.
.
.
/************ Various unrelated codes *******/
.
.
.
uint8_t frame802154_create
(frame802154_t *p, uint8_t *buf, uint8_t buf_len){
  int c;
  uint8_t *tx_frame_buffer;
  uint8_t pos;

  field_len(p, &flen);

  if(3 + flen.dest_pid_len + flen.dest_addr_len +
     flen.src_pid_len + flen.src_addr_len
     + flen.aux_sec_len > buf_len) {
    /* Too little space for headers. */
    return 0;
  }

  /* OK, now we have field lengths.  Time to actually construct */
  /* the outgoing frame, and store it in tx_frame_buffer */
  tx_frame_buffer = buf;
  tx_frame_buffer[0] = (p->fcf.frame_type & 7) |
    ((p->fcf.security_enabled & 1) << 3) |
    ((p->fcf.frame_pending & 1) << 4) |
    ((p->fcf.ack_required & 1) << 5) |
    ((p->fcf.panid_compression & 1) << 6);
  tx_frame_buffer[1] = ((p->fcf.dest_addr_mode & 3) << 2) |
    ((p->fcf.frame_version & 3) << 4) |
    ((p->fcf.src_addr_mode & 3) << 6) |
    ((packetbuf_attr(PACKETBUF_ATTR_CONGESTION) & 3));
  /* sequence number */
  tx_frame_buffer[2] = p->seq;
  pos = 3;
.
.
.
/************ Various unrelated codes *******/
.
.
.
uint8_t
frame802154_parse(uint8_t *data, uint8_t len, frame802154_t *pf)
{
  uint8_t *p;
  uint8_t c;

  if(len < 3) {
    return 0;
  }

  p = data;

  memset(&fcf, 0, sizeof(fcf));
  /* decode the FCF */
  fcf.frame_type = p[0] & 7;
  fcf.security_enabled = (p[0] >> 3) & 1;
  fcf.frame_pending = (p[0] >> 4) & 1;
  fcf.ack_required = (p[0] >> 5) & 1;
  fcf.panid_compression = (p[0] >> 6) & 1;

  fcf.dest_addr_mode = (p[1] >> 2) & 3;
  fcf.frame_version = (p[1] >> 4) & 3;
  fcf.src_addr_mode = (p[1] >> 6) & 3;

  packetbuf_set_attr(PACKETBUF_ATTR_CONGESTION, (p[1] & 3));

  /* copy fcf and seqNum */
  memcpy(&pf->fcf, &fcf, sizeof(frame802154_fcf_t));
  pf->seq = p[2];
  p += 3;                              /* Skip first three bytes */

  /* Destination address, if any */
  if(fcf.dest_addr_mode) {
    /* Destination PAN */
    pf->dest_pid = p[0] + (p[1] << 8);
    p += 2;
.
.
.
/************ Various unrelated codes *******/
.
.
.
```

It is worthy to mention that the above provided modifications are utilizing only the 2 out of the 3 available bits in the reserved field of 802.15.4 header. With a couple of extra lines *(extend the current method used to utilize 3 bits)* all 3 bits can be utilized.

Appendix F

Pseudocodes

F.1 Application Layer Pseudocodes

The application layer used in the majority of the experiments *(with small variations between test-bed and simulations)* and some of it's basic functionalities will be described in the form of pseudocode as follows:

Algorithm 1 Packet received (application)

```
 1: procedure TCPIP_HANDLER(void)
 2:     if !$congestion\_flag$ && packet contains data then
 3:         $congestion\_flag = 1$
 4:         $inactive\_flag = 0$
 5:         packet_to_send ← read from received packet
 6:         packet_transmission_interval ← read from received packet    ▷ The received packet contains all the information required for the autoconfiguration
 7:     else
 8:         $congestion\_flag = 0$
 9:         $inactive\_flag = 1$
10:         seq_id = 0
11:         unspecify the UDP connection    ▷ reset the destination address to 0
12:     end if
13:     sender_addr ← read from received packet
14: end procedure
```

Algorithm 2 Send packet (application)

```
1: procedure TIMEOUT_HANDLER($energy\_flag$)
2:     if $energy\_flag$ then        ▷ this function is used for the transmission of both normal and energy measurement packets. What packet this function will transmit depends on the energy_flag argument
3:         calculate energy measurements and use as the application packet payload
4:     else
5:         use as the application packet payload any information is related to the measured metrics such as: seq_no
6:     end if
7:     send packet to→sender_addr        ▷ there is a pre-configured, default sender_addr used by the application before the first packet reception
8: end procedure
```

Algorithm 3 In application's process thread

```
1: Applications infinite loop:
2: while 1 do
3:     PROCESS YIELD
4:     if $packet\_transmission\_interval\_timer \leftarrow expired$ then
5:         if $inactive\_flag = 0$ then
6:             timeout handler(0)
7:             if $seq\_id >= packet\_to\_send$ then
8:                 $congestion\_flag = 0$
9:                 $inactive\_flag = 1$
10:                $seq\_id = 0$
11:                unspecify the UDP connection
12:            else
13:                Start $packet\_transmission\_interval\_timer \leftarrow packet\_transmission\_interval$        ▷ Start timer with the requested interval
14:            end if
15:        end if
16:    else if $energy\_timer \leftarrow expired$ then
17:        timeout handler(1)
18:    else if received packet then
19:        tcpip handler
20:    end if
21: end while
```

F.2 Data collection server Pseudocodes

The data collection server used during the test-bed experiments and some of it's basic functionalities will be described in the form of pseudocode as follows:

Algorithm 4 Print statistics for a node

```
 1: procedure DUMP_STATS(void)
 2:     for $i = 0$, $i < MAX\_NODES$,$i++$ do
 3:         if memcmp(nodestats[i].sender,in6addr_any,size of IPv6 addres) then    ▷ checks if position is empty
 4:             return &nodestats[i]
 5:             return
 6:         else
 7:             open stats-file
 8:             if pointer_to_stats-file != NULL then
 9:                 file_print(nodestats[i].sender, nodestats[i].count)    ▷ will print the total amount of packets received by a specific IP address
10:             end if
11:         end if
12:     end for
13: end procedure
```

Algorithm 5 Save nodes in a table

```
 1: procedure ALLOCATE_NODE(struct pointer to IPv6 addres new_node)
 2:     for $i = 0$, $i < MAX\_NODES$,$i++$ do
 3:         if memcmp(nodestats[i].sender,in6addr_any,size of IPv6 addres) then    ▷ checks if position is empty
 4:             memcpy (nodestats[i].sender , new_node , size of IPv6 addres )    ▷ save the new node in the empty position
 5:         end if
 6:     end for
 7:     return 0
 8: end procedure
```

Algorithm 6 Search if a node exists in the table

```
1: procedure FIND_NODE(struct pointer to $IPv6$ addres this_node)
2:     for $i = 0$, $i < MAX\_NODES$,$i++$ do
3:         if memcmp(nodestats[i].sender,this_node,size of IPv6 addres) then ▷ search if the node is saved in the table
4:             return &nodestats[i]    ▷ return a pointer to the address that the requested node is saved
5:         end if
6:     end for
7:     return 0
8: end procedure
```

Algorithm 7 Receiving packets and collecting the data

```
1: procedure COLLECT(void)
2:     if packetisreceived then
3:         get-time-of-day
4:         node = find_node()
5:         if node then
6:             if validpacket then
7:                 node → count ++   ▷ increment the packet count for the specific node
8:             end if
9:         else if node = allocate_node() then
10:            if validpacket&& node then
11:                node → count ++   ▷ increment the packet count for the specific node
12:            end if
13:        else
14:            print error
15:        end if
16:        if packet == energy_packet then
17:            open energy-file
18:            if pointer_to_energy-file ! = NULL then
19:                file_print(energy_attributes)   ▷ will print in the file all of the required energy measurements
20:            end if
21:        else
22:            open stats-file
23:            if pointer_to_stats-file ! = NULL then
24:                file_print(transmission_attributes)   ▷ will print in the file information such as the packet payload, seq-no etc.
25:            end if
26:        end if
27:    else
28:        wrong read
29:    end if
30:    return bytes-received
31: end procedure
```

F.3 DCCC6 Pseudocodes

Some of the basic functionalities of DCCC6 are described in the form of pseudocode:

Algorithm 8 Packet received from upper layers

```
1: procedure SEND_PACKET(function pointer to sent, void pointer)
2:     increase the sequence number
3:     if packet != broadcast && no duty cycle then
4:         parent_node = destination address ▷ when multiple sinks we store the parent nodes in a table
5:     end if
6:     if packet_queue != full then
7:         if packet_type == reliable then
8:             push packet in packet queue        ▷ beginning of the queue
9:         else
10:            add packet in packet queue         ▷ end of the queue
11:        end if
12:        start transmission timer
13:    else
14:        send packet to lower layer
15:    end if
16: end procedure
```

Algorithm 9 Send congestion notification message

```
1: procedure CONGESTION_NOTIFICATION(void)
2:     clear transmission buffer
3:     if no duty cycle then
4:         destination address = broadcast address
5:     else
6:         destination address = address of node causing congestion
7:     end if
8:     congestion_notification flag = 1
9:     transmit congestion notification packet
10: end procedure
```

Algorithm 10 Reduce transmission rates when congestion notification packet is received

```
1: procedure REDUCE_RATE($void$)
2:     if $rate < max\_rate$ then        ▷ max_rate depends on the duty cycle configuration
3:         $rate = max\_rate$
4:     end if
5:     if $rate < min\_rate$ then        ▷ high value related to the duty cycle configuration
6:         $rate = var * sqrt(min\_rate)/sqrt(rate)$        ▷ var=controls the slope
7:     end if
8: end procedure
```

Algorithm 11 Increase transmission rates after a successful transmission

```
1: procedure INCREASE_RATE($void$)
2:     if $rate > min\_rate$ then        ▷ min_rate depends on the duty cycle configuration
3:         $rate = var1 * rate * sqrt(nodesthatinterfere + 1) / (var2 * sqrt(max\_rate)) - sqrt(rate)$
4:     end if
5: end procedure
```

Algorithm 12 Refresh table with interference nodes

```
1: procedure REFRESH_INTERFERENCE($void$) ▷ periodically resets the table of interfere nodes
2:     for $i = 0$,$i < table\_size$,$increase\ i$ do
3:         $interference\_table[i] = NULL$
4:     end for
5: end procedure
```

Algorithm 13 Receiving a packet

```
procedure INPUT_PACKET($void$)
    if $unicast$ && $congestion\_notification$ then
        $reduce\ rates$
    else if $broadcast$ && $congestion\_notification$ && $from\ parent\ node$ then
        $reduce\ rate$
    end if
    if $unicast$ then                                        ▷ measuring child nodes
        for $i = 0, i < max\_neighbours, increase\ i$ do
            if $source\_address == interference\_table[i]$ then
                $exit\ for\ loop$
                $flag\_found = 1$
            else
                $save\ address\ to\ the\ first\ empty\ place\ in\ interference\_table$
            end if
        end for
        if $flag\_found == 0$ then
            $interference\_counter = 0$
            for $i = 0, i < max\_neighbours, increase\ i$ do
                if $interference\_table[i]\ != NULL$ then
                    $increase\ interference\_counter\ by\ one$
                end if
            end for
        end if
        if $packet\_queue\_size > threshold\ and\ node\_childs\ != 0$ then
            $send\ congestion\ notification\ packet$
            if $threshold < max\_packet\_queue\_size$ then
                $increase\ threshold$        ▷ avoid simultaneous transmissions of congestion notification packet
            end if
        else
            if $threshold > default\_size$ then
                $decrease\ threshold$
            end if
        end if
    end if
end procedure
```

Algorithm 14 Returning the status of the packet transmission

```
procedure PACKET_SENT($void\ pointer$, $status$ , $number\_of\_transmissions$)
    if $state == ok$ then
        $break$
    else if $state == noack$ then
        $increasetransmissionsby1$
    else if $state == collision$ then
        $increasecollisionsby1$
    end if
    if $state == collision\ or\ state == noack$ then
        $time = channel\_check\_interval$
        $time = time + (random\_number\ modulo\ transmissions * time)$
        if $transmissions < max\_transmissions$ then
            $retransmit\ the\ packet\ after\ time$
        else
            $drop\ packet$
        end if
    else
        $remove\ successfully\ transmitted\ packet$
    end if
end procedure
```

F.3.1 CADC Pseudocodes

Some of the basic functionalities of DCCC6 are described in the form of pseudocode:

Algorithm 15 Reset congestion parameters

```
1: procedure RESET_PARAM(void)    ▷ reset all of the congestion parameters to their default values
2: end procedure
```

Algorithm 16 Update channel check rate and congestion parameters when forwarding state ends

```
1: procedure FORWARDING_NOW(void)
2:     reset_param()
3:     update_duty_cyle()    ▷ this function informs RDC layer for the channel check rate changes
4:     CADC_state = DEFAULT_STATE
5: end procedure
```

Algorithm 17 Reset node's channel check rate back to default when inactive

```
1: procedure IDLE_STATE(void pointer)
2:     if inactive then    ▷ flag shows activity of the node
3:         if node is not inactive due to misconfiguration then
4:             channel_check_rate=DEFAULT_MINCCR
5:             update_duty_cyle()    ▷ this function informs RDC layer for the channel check rate changes
6:         else
7:             reset_param()
8:         end if
9:     end if
10:    inactive=1
11:    reset function timer    ▷ this function is executed periodically by a timer
12: end procedure
```

Algorithm 18 Packet is received

```
 1: procedure INPUT_PACKET(void)
 2:     if !broadcast then
 3:         inactive=0
 4:         in_traffic++
 5:         broadcast_flag=0
 6:         if packet_attribute_congestion then
 7:             CCR_CMP()
 8:         end if
 9:     else
10:         broadcast_flag=1
11:     end if
12:     forward packet to the upper layers
13: end procedure
```

Algorithm 19 Compares node's channel check rate with the one received from child nodes

```
 1: procedure CCR_CMP(void)
 2:     Switchpacket_attribute_congestion
 3:     case 1:CCR_received=4
 4:     case 1:CCR_received=8
 5:     case 1:CCR_received=16
 6:     case 4:CCR_received=32
 7:     case 5:CCR_received=64
 8:     EndSwitch
 9:     if CCR_received ¿ highest_CCR_received then
10:         highest_CCR_received=CCR_received
11:     end if
12:     if CCR_received ¿ Current_CCR then
13:         Current_CCR=CCR_received
14:         Current_state=FORWARDING_STATE
15:         start timer for the execution of forwarding_now function
16:     end if
17: end procedure
```

Algorithm 20 Transmit an unicast packet

```
 1: procedure TRANSMIT_QUEUED_PACKET(void)
 2:     if RDC is transmitting then
 3:         return
 4:     end if
 5:     Switch Current_CCR
 6:     case 4: packet_attribute_congestion=1
 7:     case 8: packet_attribute_congestion=2
 8:     case 16: packet_attribute_congestion=3
 9:     case 32: packet_attribute_congestion=4
10:     case 64: packet_attribute_congestion=5
11:     EndSwitch
12:     forward packet to the lower layers
13: end procedure
```

Algorithm 21 CADC state decision

```
 1: procedure NODE_STATE(void)
 2:     if packet_queue >= hi_threshold then
 3:         if succesfull_transmissions <= fail_threshold && Current_CCR <=
    MAX_CCR_COLLAPSE then
 4:             state = COLLAPSE_STATE
 5:         else
 6:             state = CONGESTION_STATE
 7:         end if
 8:     else if packet_queue >= low_threshold then
 9:         if succesfull_transmissions > in_traffic then
10:             state = NORMAL_STATE
11:         else
12:             state = CONGESTION_STATE
13:         end if
14:     else
15:         if succesfull_transmissions > in_traffic && fail_transmissions <=
    fail_threshold && Current_CCR > MIN_CCR then
16:             state = OVER_DC_STATE
17:         else
18:             state = NORMAL_STATE
19:         end if
20:     end if
21:     reset_param()
22:     calculate_CCR()
23: end procedure
```

Algorithm 22 Calculate the new channel check rate based on the state in which the node is

```
 1: procedure CALCULATE_CCR(void)
 2:     Switch state
 3:     case COLLAPSE_STATE:
 4:     Current_CCR = Current_CCR << 2        ▷ quadruples channel check rate
 5:     state = FORWARDING_STATE
 6:     start timer for the execution of forwarding_now() function
 7:     case CONGESTION_STATE:
 8:     if Current_CCR < MAX_CCR then
 9:         Current_CCR = Current_CCR << 1        ▷ doubles channel check rate
10:     end if
11:     state = FORWARDING_STATE
12:     start timer for the execution of forwarding_now() function
13:     case OVER_DC_STATE:
14:     if Current_CCR ¿ highest_CCR_received then ▷ cannot reduce more than
    the child's channel check rate
15:         Current_CCR = Current_CCR >> 1
16:         update_duty_cyle()                    ▷ halves channel check rate
17:     end if
18:     case NORMAL_STATE:
19:     Node operates at the optimal channel check rate. Do nothing
20:     EndSwitch
21: end procedure
```

Algorithm 23 Retrieving the status of a packet's transmission

```
procedure PACKET_SENT(void pointer, integer status, integer number of transmissions)
    Switch status
    case OK:
    succesfull_transmissions++
    case NO_ACK:
    not_acknowledged++
    case COLLISION:
    collisions++
    EndSwitch
    total_tranmissions = (collisions + not_acknowledged + succesfull_transmissions)
    if total_tranmissions >= NUMBER_OF_STATECHECK_TRANSMISSIONS && Current_STATE != FORWARDING_STATE then
        node_state()
    end if
    if (status == COLLISION || status == NO_ACK) && packet → transmissions > packet → MAX_transmissions then
        drop packet
    else if status == COLLISION then
        time = get_time()  ▷ the value returned depends on the channel check rate
        packet → transmissions++
        if broadcast_flag then  ▷ broadcast transmissions have always the maximum transmission duration and thus we should always wait longer
            time = BC_TIME + (random_time % (packet → transmissions * time))
            broadcast_flag = 0
        else
            time = time + (random_time % (packet → transmissions * time))
            set packet retransmission timer
        end if
    else if status == NO_ACK then
        if Current_STATE == FORWARDING_STATE then
            retransmit packet now  ▷ stitching technique
        else
            packet → transmissions++
            random_time % (packet → transmissions * get_time())  ▷ when no acknowledgment retransmission time window can start from 0
        end if
        set packet retransmission timer
    else
        drop packet
    end if
end procedure
```

Appendix G

Logs of CADC's Validation Through Simulation Experiments

In this section, log files from the simulation experiments used for validation of CADC are presented. The numbers in the first column represent the time in μSec. The second column shows the Node's ID *(through that we can see what actions each node performed at each time of the simulation)*. The third column contains the message printed by each node. Nodes will periodically print their queue status as well as the rest of the parameters used by CADC. Figure G.1 illustrates the setup of CADC evaluation experiments in cooja simulator.

The log file of an experiment with low congestion *(pact transmission interval set at 250 ms)* follows:

```
496             ID:2    Rime started with address 0.18.116.2.0.2.2.2
507             ID:2    MAC 00:12:74:02:00:02:02:02
                        Contiki-2.5-1700-g350296a started.
                        Node id is set to 2.
518             ID:2    CADC CADC-RDC, channel check rate 8 Hz,
                        radio channel 26
537             ID:2    Tentative link-local IPv6 address:
                        fe80:0000:0000:0000:0212:7402:0002:0202
539             ID:2    Starting 'UDP client process'
642             ID:1    Rime started with address 0.18.116.1.0.1.1.1
653             ID:1    MAC 00:12:74:01:00:01:01:01
                        Contiki-2.5-1700-g350296a started.
                         Node id is set to 1.
664             ID:1    CADC CADC-RDC, channel check rate 8 Hz,
                        radio channel 26
683             ID:1    Tentative link-local IPv6 address:
                        fe80:0000:0000:0000:0212:7401:0001:0101
685             ID:1    Starting 'UDP server process'
689             ID:1    Conf: CADC 8 16  CADC-RDC
1159            ID:3    Rime started with address 0.18.116.3.0.3.3.3
1170            ID:3    MAC 00:12:74:03:00:03:03:03
                        Contiki-2.5-1700-g350296a started.
                        Node id is set to 3.
1182            ID:3    CADC CADC-RDC, channel check rate 8 Hz,
                         radio channel 26
1200            ID:3    Tentative link-local IPv6 address:
```

```
                    fe80:0000:0000:0000:0212:7403:0003:0303
1203            ID:3    Starting 'UDP client process'
22519   ID:3    NORMAL! Qth below levels of congestion
22533   ID:2    NORMAL! Qth below levels of congestion
23897   ID:3    NORMAL! Qth below levels of congestion
23911   ID:2    NORMAL! Qth below levels of congestion
25272   ID:3    NORMAL! Qth below levels of congestion
25286   ID:2    NORMAL! Qth below levels of congestion
26644   ID:3    NORMAL! Qth below levels of congestion
26658   ID:2    NORMAL! Qth below levels of congestion
28022   ID:3    NORMAL! Qth below levels of congestion
28036   ID:2    NORMAL! Qth below levels of congestion
29503   ID:2    NORMAL! Qth below levels of congestion
29519   ID:3    NORMAL! Qth below levels of congestion
31261   ID:2    NORMAL! Qth below levels of congestion
32021   ID:3    NORMAL! Qth below levels of congestion
33207   ID:2    CONGESTION! queue len: 8 Rsucc: 4 /2
33704   ID:2    duty cycle updated to: 16
33723   ID:1    CCR received: 16 higher than operating CCR: 8
34208   ID:3    NORMAL! Qth below levels of congestion
34224   ID:1    duty cycle updated to: 16
34474   ID:2    NORMAL!  from congestion Ql > 1 Rserv >1
35378   ID:2    CONGESTION! queue len: 8 Rsucc: 5/2  /* Rserv = 5/(10-5)
                Rfail = 5/2*/*/
35551   ID:1    CCR received: 32 higher than operating CCR: 16
35647   ID:3    NORMAL! Qth below levels of congestion
35875   ID:2    duty cycle updated to: 32
36053   ID:1    duty cycle updated to: 32
36307   ID:2    OVERDC! queue len: 4 Rsucc: 9/2  /* Rserv = 9/(10-9)*/
36309   ID:2    duty cycle updated to : 16
37152   ID:2    NORMAL! Qth below levels of congestion
37365   ID:3    NORMAL! Qth below levels of congestion
38567   ID:2    OVERDC! queue len: 1 Rsucc: 10/2  /* Rserv > 1 by default*/
38568   ID:2    duty cycle updated to : 8
39429   ID:3    NORMAL! Qth below levels of congestion
40793   ID:2    CONGESTION! queue len: 6 Rsucc: 4/2  /* Rserv = 4/(10-4)*/
41289   ID:2    duty cycle updated to: 16
41554   ID:3    NORMAL! Qth below levels of congestion
42196   ID:2    NORMAL!  from congestion Ql > 1 Rserv >1
43042   ID:2    NORMAL! Qth below levels of congestion
43615   ID:3    NORMAL! Qth below levels of congestion
43954   ID:2    NORMAL! Qth below levels of congestion
44863   ID:2    NORMAL! Qth below levels of congestion
45615   ID:3    NORMAL! Qth below levels of congestion
45811   ID:2    NORMAL! Qth below levels of congestion
47192   ID:2    OVERDC! queue len: 1 Rsucc: 9/2  /* Rserv = 9/(10-9)*/
47193   ID:2    duty cycle updated to : 8
47676   ID:3    NORMAL! Qth below levels of congestion
49292   ID:2    NORMAL! Qth below levels of congestion
49802   ID:3    NORMAL! Qth below levels of congestion
51041   ID:2    CONGESTION! queue len: 7 Rsucc: 5/2  /* Rserv = 5/(10-5)
                Rfail = 5/2*/
51539   ID:2    duty cycle updated to: 16
51990   ID:3    NORMAL! Qth below levels of congestion
52478   ID:2    NORMAL!  from congestion Ql > 1 Rserv >1
53298   ID:2    NORMAL!  from congestion Ql > 1 Rserv >1
53929   ID:3    NORMAL! Qth below levels of congestion
54196   ID:2    NORMAL! Qth below levels of congestion
55042   ID:2    NORMAL! Qth below levels of congestion
58516   ID:2    node is idel CCR returned to default: 8
60662   ID:1    node is idel CCR returned to default: 8
```

The log file of an experiment with heavy congestion *(pact transmission interval set at 62.5 ms)* follows:

```
496             ID:2    Rime started with address 0.18.116.2.0.2.2.2
507             ID:2    MAC 00:12:74:02:00:02:02:02
                        Contiki-2.5-1700-g350296a started.
                        Node id is set to 2.
518             ID:2    CADC CADC-RDC, channel check rate 8 Hz,
                        radio channel 26
537             ID:2    Tentative link-local IPv6 address:
                         fe80:0000:0000:0000:0212:7402:0002:0202
539             ID:2    Starting 'UDP client process'
642             ID:1    Rime started with address 0.18.116.1.0.1.1.1
653             ID:1    MAC 00:12:74:01:00:01:01:01
                        Contiki-2.5-1700-g350296a started.
                        Node id is set to 1.
664             ID:1    CADC CADC-RDC, channel check rate 8 Hz,
                         radio channel 26
683             ID:1    Tentative link-local IPv6 address:
                         fe80:0000:0000:0000:0212:7401:0001:0101
685             ID:1    Starting 'UDP server process'
689             ID:1    Conf: CADC 8 16 CADC-RDC
1159            ID:3    Rime started with address 0.18.116.3.0.3.3.3
1170            ID:3    MAC 00:12:74:03:00:03:03:03
                        Contiki-2.5-1700-g350296a started.
                         Node id is set to 3.
1182            ID:3    CADC CADC-RDC, channel check rate 8 Hz,
                        radio channel 26
1200            ID:3    Tentative link-local IPv6 address:
                         fe80:0000:0000:0000:0212:7403:0003:0303
1203            ID:3    Starting 'UDP client process'
22923   ID:2    NORMAL! Qth below levels of congestion
23149   ID:3    CONGESTION! queue len: 7 Rsucc: 10 /2
23399   ID:2    CCR received: 16 higher than operating CCR: 8
23464   ID:1    CCR received: 16 higher than operating CCR: 8
23653   ID:3    duty cycle updated to: 16
23901   ID:2    duty cycle updated to: 16
23967   ID:1    duty cycle updated to: 16
24839   ID:2    CONGESTION! queue len: 8 Rsucc: 3/2  /* Rserv = 3/(10-3)*/
24957   ID:1    CCR received: 32 higher than operating CCR: 16
25022   ID:3    COLLAPSE! queue len: 10 Rsucc: 2/2  /* Rserv = 2/(10-2)*/
25151   ID:2    CCR received: 64 higher than operating CCR: 32
25418   ID:1    CCR received: 64 higher than operating CCR: 32
25528   ID:3    duty cycle updated to: 64
25649   ID:2    duty cycle updated to: 64
25893   ID:3    NORMAL!  from congestion Ql > 1 Rserv >1
25916   ID:2    NORMAL!  from congestion Ql > 1 Rserv >1
25920   ID:1    duty cycle updated to: 64
26152   ID:2    CONGESTION! queue len: 8 Rsucc: 5/2  /* Rserv = 5/(10-5)*/
26155   ID:2    NORMAL STEP2!  already at max CCR
26195   ID:3    NORMAL! Qth below levels of congestion
26396   ID:2    CONGESTION! queue len: 8 Rsucc: 5/2  /* Rserv = 5/(10-5)*/
26399   ID:2    NORMAL STEP2!  already at max CCR
26488   ID:3    NORMAL! Qth below levels of congestion
26646   ID:2    NORMAL!  from congestion Ql > 1 Rserv >1
26901   ID:2    NORMAL! Qth below levels of congestion
27000   ID:3    NORMAL! Qth below levels of congestion
27244   ID:2    NORMAL! Qth below levels of congestion
27258   ID:3    NORMAL! Qth below levels of congestion
27473   ID:2    NORMAL! Qth below levels of congestion
27704   ID:2    NORMAL! Qth below levels of congestion
27868   ID:3    OVERDC! queue len: 1 Rsucc: 10/2  /* Rserv >1 by default*/
27870   ID:3    duty cycle updated to : 32
```

```
27932   ID:2    NORMAL! Qth below levels of congestion
28318   ID:2    OVERDC! queue len: 1 Rsucc: 9/2  /* Rserv = 9/(10-9)*/
28319   ID:2    duty cycle updated to : 32
28569   ID:3    OVERDC! queue len: 1 Rsucc: 10/2  /* Rserv >1 by default*/
28570   ID:3    duty cycle updated to : 16
28990   ID:2    OVERDC! queue len: 1 Rsucc: 10/2  /* Rserv >1 by default*/
28991   ID:2    duty cycle updated to : 16
29287   ID:3    NORMAL! Qth below levels of congestion
30065   ID:2    NORMAL! Qth below levels of congestion
30601   ID:3    OVERDC! queue len: 3 Rsucc: 10/2  /* Rserv >1 by default*/
30602   ID:3    duty cycle updated to : 8
30911   ID:2    NORMAL! Qth below levels of congestion
31845   ID:2    NORMAL! Qth below levels of congestion
32660   ID:2    NORMAL! Qth below levels of congestion
32664   ID:3    COLLAPSE! queue len: 10 Rsucc: 2/2  /* Rserv = 2/(10-2)*/
32853   ID:2    CCR received: 32 higher than operating CCR: 16
33231   ID:3    duty cycle updated to: 32
33352   ID:2    duty cycle updated to: 32
34006   ID:3    CONGESTION! from collapse Queue_threshold > high && Rsucc >1 /
34073   ID:2    CCR received: 64 higher than operating CCR: 32
34513   ID:3    duty cycle updated to: 64
34571   ID:2    duty cycle updated to: 64
34851   ID:3    NORMAL!  from congestion Ql > 1 Rserv >1
35036   ID:2    NORMAL!from over dc received CCR is higher can't reduce
35148   ID:3    NORMAL! Qth below levels of congestion
35503   ID:3    CONGESTION! queue len: 6 Rsucc: 4/2  /* Rserv = 4/(10-4)*/
35506   ID:3    NORMAL STEP2!  already at max CCR
35867   ID:3    NORMAL!  from congestion Ql > 1 Rserv >1
36068   ID:2    NORMAL!from over dc received CCR is higher can't reduce
36143   ID:3    NORMAL! Qth below levels of congestion
36571   ID:3    OVERDC! queue len: 1 Rsucc: 10/2  /* Rserv >1 by default*/
36573   ID:3    duty cycle updated to : 32
36852   ID:3    OVERDC! queue len: 1 Rsucc: 10/2  /* Rserv >1 by default*/
37018   ID:2    duty cycle updated to : 32
37020   ID:2    NORMAL! Qth below levels of congestion
37211   ID:3    NORMAL! Qth below levels of congestion
37742   ID:3    NORMAL! Qth below levels of congestion
40525   ID:2    node is idle CCR returned to default: 8
41618   ID:3    node is idle CCR returned to default: 8
42662   ID:1    node is idle CCR returned to default: 8
```

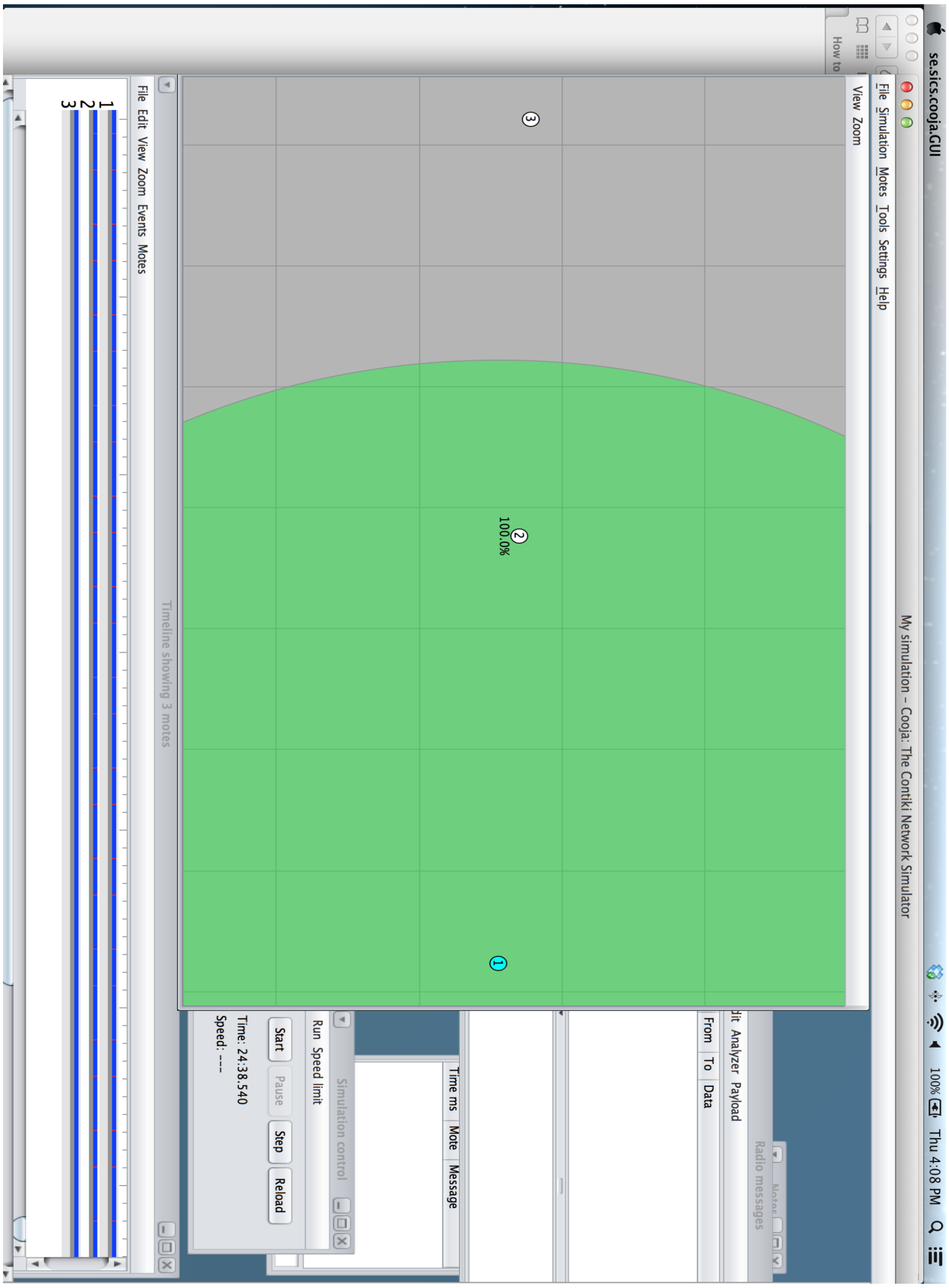

Figure G.1: Cooja simulator CADC evaluation experiments