

IoT-Based Sensor Networks:
Architectural Organization, Virtualization
and Network Re-orchestration

Indrajit Shankar Acharyya

A thesis submitted to
Auckland University of Technology
in fulfilment of the requirements for the degree of
Doctor of Philosophy (PhD)

2021

School of Engineering, Computer & Mathematical Sciences

Attestation of Authorship

“I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgments), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.”

Indrajit Shankar Acharyya:

Date: 19-04-2021

Summary

Internet of ‘Things’ (IoT), an extension of localised ‘Wireless’ Sensor Networks (WSN), has been employed to realize a multitude of smart, intelligent and pervasive Cyber Physical System (CPS) infrastructures. CPS encompasses a host of technological and architectural challenges like low-power communication, protocol conversions, data transport and the ability to interoperate with other IoT technologies. This makes CPS significantly complex and reduces its flexibility to adapt. A typical IoT-based sensor network may to a certain extent, lack key softwarization-enabled operational drivers that may introduce significant constraints on its ability to flexibly engage with its external surroundings.

Flexible re-orchestration of such complex IoT based sensor networks, however, is vital towards aligning system ‘dynamics’ with that of a monitored ‘physical’ phenomenon while operating in dynamic physical environments (for example WSNs deployed for outdoor applications such as forest fire monitoring). Arguably, considerable levels of operational flexibility can be achieved through a cloud-based architectural framework that hosts the required operating tools to allow for software-defined network virtualization that enable suitable re-orchestrations of the related physical network.

In a nutshell, research work documented within this thesis endeavours towards rendering a sensor network capable of undergoing desired flexible re-orchestrations via converging upon a novel architectural proposition inclusive of modularization, cloud-based virtualization, software control via ‘command-driven reconfigurability’, and maintenance of a library for additional firmware modules (for each of the nodes at the physical level), among others. Other equally noteworthy and innovative contributions of this thesis pertain to outlining of a seemingly logical strategy for the sensor network ‘re-orchestration’ process (that spans across ‘three’ phases of, ‘Data Analysis and Event-Identification’, ‘Re-orchestration-Planning’ and ‘Re-orchestration-Execution’) as well as both determining and formulating a generic model for the latency associated with the same.

The approach adopted herein is to allow for the underlying physical layer to undergo desired node and network-level (including topological) re-orchestrations (based on the outcomes derived from the cloud) in a flexible and expeditious manner during run-time through a ‘Command-driven’ re-configurability approach. This relatively simplistic yet expedient approach involves loading of a ‘unified firmware’ (i.e., one encompassing the requisite, ‘well-defined’ software modules) onto nodes (assumed to be capable of accommodating for and executing the corresponding functional roles owing to the enhanced capabilities ushered in by the advancements attained in the field of SoC and Embedded Systems technologies) to allow for conditional execution of the same remotely by means of ‘commands’. In order to augment the flexibilities that could be offloaded by the node over time based on the service requirements, a library of ‘reusable firmware modules’ (within which the requisite new functional modules could be integrated from time-to-time) could be maintained to be readily accessible by the main firmware.

In regard to the above context, it is deemed worthy to reiterate that the thesis underscores the key prerequisites for the above prior to laying the concept in chapter four. Firstly, this includes identifying and clearly defining the core functional components (constituting any IoT-

based sensor network organization viz., ‘leaf’, router and ‘Gateway’ functionalities) as ‘modules. The second prerequisite pertains to modularization of the core functional components that have been identified and defined. Virtualization of the core functional modules so identified and thereby the entire network (essentially, cloud-level Network Virtualization i.e., ‘NV’) that ‘logically’ (i.e., from a software standpoint) mimics the operational dynamics of the underlying physical network functions will form the third prerequisite. As alluded to earlier, the fourth prerequisite refers to the library of reusable ‘firmware modules’ at the node level (for augmented flexibility).

The thesis is sectioned into seven different chapters, each accounting for a specific element of the overall work. The first chapter provides an overview of the various technological domains and aspects associated with this research work, whilst laying out the necessary background, vision and motivation behind the same. The second chapter accounts for a review of the existing literature pertaining to the various elements associated with this research viz., WSN virtualization, softwarization, re-orchestration and associated network downtime (as well as other architectural frameworks designed with relatively similar motives in mind). Information pertaining to the tools employed for virtualization and hardware implementation purposes are provided in the third chapter. As elaborated above, Chapter 4 firstly spells out the key prerequisites for the proposed architecture prior to describing the same, along with its internal components. It then outlines the strategy adopted for the re-orchestration process, including formulation of a generic model for the latency that the network may experience as a result of the same. By means of certain pertinent example cases of software-defined sensor network re-orchestrations, chapter 5 details the specifics of both virtual and physical implementations, conducted via utilizing the Contiki-oriented virtual platform of the Cooja simulator as well as the Contiki-ported Texas Instruments CC2538 wireless transceivers respectively. It also brings to the fore the practicability of employing Contiki as a tool for software development that allows for precise replication of the codes employed for physical motes at the virtual level, whilst leveraging on the same to better analyse and conduct more accurate performance evaluations pertaining to the re-orchestration process. As a means to demonstrate the workability of the proposed concept with respect to a real-life scenario, chapter 6 deals with the use case pertaining to forest fire monitoring wherein dynamic re-orchestration of sensor network so deployed could significantly aid (pre-emptive) re-routing of network dataflow and/or maintenance of network connectivity in the event of network fragmentation emanating out of rapidly spreading uncontained fire outbreaks. Chapter 7 puts forth the conclusion of this thesis work, along with the future course of work to be undertaken.

Acknowledgments

I would like to begin this by expressing my deep regards and profound gratitude to Prof. Adnan Al-Anbuky, whose adroit guidance, coupled with a deep penchant to incessantly innovate and improvise, inspired me to persevere relentlessly during the course of my doctoral journey. His eclectic mentorship, especially the thought-provoking meeting sessions were highly motivational, propelling me to think ‘out-of-the-box’. Such supervision tends to go a long way towards both inspiring and helping one to ‘cut the mustard’, despite all the hurdles and challenges involved. I consider myself to be privileged to have been seamlessly groomed as a ‘researcher’ under his tutelage, right from the onset of my doctoral journey. I truly am immensely indebted to him for providing me with this excellent opportunity to be a part of the *SeNSe* research team wherein I was provided with the necessary means and guidance to help publish my works enrich my knowledge base in the process.

A PhD journey may at times prove to be a demanding one, one that can seldom be traversed without the helping hands offered by certain key selfless individuals along the way. Fortunately, such kind, noble-hearted and thoroughly professional people have been a part of this journey of mine right from the very start and have massively contributed towards unlocking my true potential. I wish to extend my heartfelt thanks to my secondary supervisor, Dr. Sivakumar Sivaramakrishnan for offering dedicated and selfless help with the technical intricacies as well as the conceptual aspects pertaining to my research. Deconstructing the complex concepts into simpler, lucid ones were one of his key fortes, enabling me to better grasp and incorporate them within my research work. While the brainstorming sessions were of immense benefit, he also imparted motivational words of advice, significantly galvanizing my confidence on numerous occasions.

It is impossible for me go about writing this trail of acknowledgement without mentioning one Mr. Craig Walker, who over the past several years, has made an indelible contribution towards strengthening my professionalism and as well as a making a profound impact on me as an individual. Vastly knowledgeable and experienced, he has extended genuine support, magnanimity and selfless guidance without which much of my doctoral journey would simply have not come to fruition. I consider myself to be extremely fortunate to have got the opportunity to work under his mentorship and benefit from his fine sense of judgement, pragmatic approach to problem-solving as well as an uncanny knack for improvisation, certain key qualities that have been instrumental in shaping my professional and academic career alike.

I owe a special thanks to my colleague Syarifah Ezdiani who extended selfless cooperation and was always a joy to collaborate with. The doctoral journey at AUT also presented me with several opportunities to teach postgraduate /masters’ students and on one occasion guide undergraduate students. These experiences were extremely enriching and filled me with a certain amount of satisfaction and pride as I was able to do my bit to help others, albeit in a small way. I also express my sheer gratitude to the entire team of ‘iMonitor Ltd.’, which accepted me with nothing but open arms and provided me with the opportunity to gain valuable ‘industry’ work experience whilst working towards completion of my PhD study.

Finally, words cannot express my sheer indebtedness and profound gratitude towards my parents *Chitra* and *Shankar Acharyya* for their countless blessings, indelible faith and constant encouragement, that have propelled me immeasurably during this demanding phase of my

student life. Besides inculcating in me the novel human values of integrity, discipline and honesty, the limitless patience and unbreakable resolve demonstrated by them under the most trying of times and circumstances have served as an inspiration for me to remain resolute in the face of adverse or challenging circumstances that life is known to hurl at one. This thesis is lovingly dedicated to them.

A number of other individuals, whom I have not named here, have offered support to me during my doctoral journey. I wholeheartedly express my sincere gratitude to them.

Contents

Attestation of Authorship.....	i
Summary.....	ii
Acknowledgments.....	iv
List of Figures.....	x
List of Tables.....	xv
List of Abbreviations.....	xvi
Chapter 1.....	1
1.1 Introduction.....	1
1.2 ‘Wireless Sensor Networks’.....	1
1.3 Typical WSN Implementations (Structural Standpoint).....	2
1.4 Motivation for Driving Towards a Dynamically Flexible WSN.....	5
1.5 Research Questions, Aims & Objectives.....	8
1.6 Thesis Organization.....	9
1.7 Research Outcomes - Publications Based on the Thesis Work.....	9
1.8 Conclusion.....	10
Chapter 2.....	12
2.1 Introduction.....	12
2.2 WSN Virtualization.....	12
2.2.1 ‘Node-Level’ WSN Virtualization.....	13
2.2.2 ‘Network-Level’ WSN-Virtualization.....	14
2.3 Viability of WSN ‘Virtualization’.....	14
2.4 WSN Softwarization.....	16
2.4.1 SDWSN Node-Level Re-orchestrations.....	16
2.4.2 SDWSN Topological Re-orchestration.....	17
2.4.3 SDWSN Architectural Frameworks.....	18
2.4.4 NFV-WSN Merger.....	20
2.5 Re-orchestration Latency.....	21
2.6 Conclusion.....	21
Chapter 3.....	23
3.1 Introduction.....	23
3.2 Research Methods.....	23
3.3 General Organization of the Test System.....	24
3.3.1 Physical WSN Implementation.....	24
3.3.2 Remote Cloud Server Components and Associated Implementation.....	25
3.4 Contiki OS.....	28
3.4.1 Comparison of Contiki with other Operating Systems.....	29
3.4.2 Contiki Netstack.....	30
3.5 Texas Instruments CC2538 Target Hardware.....	31

3.5.1	Power Modes and Current Consumption Profile	32
3.5.2	Contiki as a Tool for TI CC2538 Configuration.....	33
3.6	Raspberry Pi SoC	34
3.7	Contiki-based Cooja as a Virtualization Platform.....	35
3.7.1	Node Mobility	36
3.7.2	In-built RSSI Model.....	36
3.7.3	‘Power-trace’ Feature.....	39
3.7.4	Provisioning for Virtual 6LoWPAN-Based Network.....	40
3.7.5	Scalability	43
3.7.6	Limitations or Challenges Associated with Cooja as a Virtualization Platform.....	43
3.8	Conclusion.....	44
Chapter 4	45
4.1	Introduction	45
4.2	WSN Softwarization: Key Pre-requisites.....	45
4.3	Key Modular Components for Flexible WSN.....	46
4.3.1	WSN Leaf Function	47
4.3.2	WSN Routing Function.....	50
4.3.3	WSN Gateway Function	51
4.4	WSN Re-orchestration	52
4.4.1	Simple Network Re-orchestration: Topological Flexibility	52
4.4.2	Example Network Topological Re-Orchestration Towards Improved Network Performance	53
4.5	SDWSN Design Approach.....	55
4.5.1	Re-configurability-based Approach: Realization via Unified Firmware.....	55
4.5.2	Library of Software Modules: Towards Incorporation of Additional Functions.....	58
4.5.3	Virtualization: Tool for Running and Testing Soft-Re-orchestration Trials	59
4.6	Proposed System Architectural Organization	59
4.6.1	Physical Layer.....	59
4.6.2	Cloud Layer	60
4.7	Strategy for Network Re-orchestration	61
4.7.1	Three-Phase WSN Re-orchestration Strategy.....	62
4.7.2	Formulation of Generic Model for WSN Re-orchestration Latency	63
4.8	Conclusion.....	66
Chapter 5	67
5.1	Introduction	67
5.2	Contiki Tool for WSN Softwarization Pre-requisites	67
5.3	Contiki-Based Pseudo Codes for Key Modular WSN Components	67
5.3.1	Contiki-Based Pseudo Code for WSN Leaf Function	67
5.3.2	Contiki-Based Pseudo Code for WSN ‘Routing’ Function	71
5.3.3	Contiki-Based Pseudo Code for WSN Gateway Function	74

5.4	Node and Network-Level WSN Re-orchestrations	77
5.4.1	Integrated (Contiki-Based) Pseudo Code.....	77
5.4.2	Sensor Selection.....	82
5.4.3	Buffer Size	83
5.4.4	Data Communication Rate	83
5.4.5	‘Arrival’ and ‘Service’ Rates	85
5.4.6	Radio Transmission Power	88
5.4.7	Channel Access Method	89
5.4.8	Channel Allocation	93
5.4.9	Network (Topology)-Operational (Functional) Re-orchestrations	94
5.5	Example WSN System Implementation.....	95
5.5.1	Physical Implementation.....	95
5.5.2	Implementation of the Virtual Environment Within the Remote Server.....	97
5.6	Example WSN Re-orchestration Scenarios.....	99
5.6.1	Simple Network (Function) Manipulation: Function Swapping Case.....	99
5.6.2	Demand for Flexibility: Topology Related-Case.....	101
5.7	Example Network Re-orchestration Scenarios with Focus on Re-orchestration Latency.....	102
5.7.1	Simple Network Manipulation: Function swapping case	102
5.7.2	Network Manipulation: Router Replacement Case	105
5.8	Conclusion.....	113
Chapter 6	114
6.1	Introduction	114
6.2	Demand for Dynamic Re-orchestration: Setting the Scene for the Example Case of Network Fragmentation in Consideration.....	114
6.3	Formulation of Fitness Model	116
6.3.1	Description of Election Parameters	116
6.3.2	Sequence of Messages for Electing a New Router	119
6.3.3	Model Formulated Towards Determining the Fitness Value of Participant Nodes	121
6.3.4	Implementation on Virtual Platform and Results	122
6.4	Conclusion.....	124
Chapter 7	125
7.1	Introduction	125
7.2	Conclusion.....	126
7.3	Future work	127
7.3.1	Realization and Incorporation of the Aspect of Digital Twin.....	127
7.3.2	Re-programmability-based Approaches	128
7.3.3	Edge-Computing Aspect.....	128

References..... 129
Appendix..... 139

List of Figures

Figure 1-1 Underlying structural approach adopted for practical WSN implementations.	2
Figure 1-2 Star topology-based WSN implementation for precision agriculture-based application.	3
Figure 1-3 Tree topology based WSN implementation for forest fire detection.	4
Figure 1-4 An example of a mesh-topology based WSN with certain 6LoWPAN enabled sensor leaf nodes.	5
Figure 1-5 Scope for flexible re-orchestration across multiple levels within a WSN.	6
Figure 1-6 Certain example topologies an IoT-based sensor network could be re-orchestrated to by means of software control: (a) ‘Star’ network ‘topology’; (b) ‘Tree’ network ‘topology’; (c) ‘Mesh’ network ‘topology’; (d) ‘Multi-hop’ network ‘topology’.....	7
Figure 3-1 General organization of the test ‘system’.....	24
Figure 3-2 Physical implementation of a 9-node TI CC2538-based sensor network within our laboratory premises.....	25
Figure 3-3 ‘Block diagram’ depicting Remote server implementation and the inter-relation among its various components.....	26
Figure 3-4 Virtual implementation of the physical 9-node TI CC2538-based sensor network within the Contiki-Cooja simulator.	27
Figure 3-5 Graphical trend of ‘sensor’ data (ambient light, temperature and RSSI) captured by the 8 leaf nodes, retrieved from Server database.	27
Figure 3-6 Contiki Netstack depicting protocols available across the various layers.	30
Figure 3-7 CC2538 Evaluation Module.....	31
Figure 3-8 SmartRF 06 Evaluation Board.	32
Figure 3-9 Current consumption profile of TI CC2538 hardware obtained via oscilloscope.	33
Figure 3-10 Raspberry Pi SoC.	35
Figure 3-11 Radio propagation model feature within Cooja (a) Network window within Cooja wherein the mobile router and the stationary leaf nodes are initially in close proximity of each other; (b) ‘Mote Interface Viewer’ window within Cooja wherein the RSSI values of the mobile router with respect to the leaf node can be viewed (within the ‘Radio’ option in its dropdown list).	37
Figure 3-12 Radio propagation model feature within Cooja (a) Network window within Cooja wherein the mobile router has moved away from the stationary leaf node to a certain extent; (b) ‘Mote Interface Viewer’ window within Cooja wherein the degraded RSSI value of the mobile router with respect to the leaf node (as compared to the earlier case) can be viewed (within the ‘Radio’ option in its dropdown list).	37
Figure 3-13 Radio propagation model feature within Cooja (a) Network window within Cooja wherein the mobile router has moved quite far away from the stationary leaf node but is still within its communication range; (b) ‘Mote Interface Viewer’ window within Cooja wherein the further degraded RSSI value of the mobile router with respect to the leaf node (as compared to the earlier case) can be viewed (within the ‘Radio’ option in its dropdown list).	38

Figure 3-14 Radio propagation model feature within Cooja (a) Network window within Cooja wherein the mobile router has moved beyond the communication range of the stationary leaf node resulting in loss of connectivity; (b) ‘Mote Interface Viewer’ window within Cooja wherein RSSI value (of the mobile router with respect to the leaf node) pertaining this particular case of loss of connectivity is reflected by an RSSI value of -100dBm (as viewed within the ‘Radio’ option in its dropdown list).....	38
Figure 3-15 Powertrace feature within Cooja (a) Network window within Cooja showing a ‘star topology’ -based 5-node virtual network within Cooja operating under CSMA mode and (b) Screenshot of the ‘Powertracker’ window showing extremely considerably high ‘Radio On(%)’, ‘Radio RX(%)’ and ‘Radio TX(%)’ values.	39
Figure 3-16 Powertrace feature within Cooja (a) Network window within Cooja showing the same ‘star topology’ -based 5-node virtual network within Cooja operating under TDMA mode and (b) Screenshot of the ‘Powertracker’ window showing extremely minimal ‘Radio On(%)’, ‘Radio RX(%)’ and ‘Radio TX(%)’ values.	40
Figure 3-17 6LoWPAN-based virtual network consisting of one border router node and 4 ‘Example server’ nodes created within Cooja.	40
Figure 3-18 Mote output window of the 5-node 6LoWPAN-based virtual network within Cooja showing the addresses of the constituent nodes are getting printed.	41
Figure 3-19 Terminal window within Contiki wherein the server IPv6 address is getting printed.	41
Figure 3-20 Terminal window within Contiki wherein the server IPv6 address is pinged to ascertain successful establishment of connection with the server.	42
Figure 3-21 6LoWPAN analyser within Contiki wherein details associated with the 6LoWPAN packets being transmitted by the nodes can be viewed.....	42
Figure 3-22 Upon entering the server address within the Mozilla Firefox web browser, details pertaining to neighbours and routes can be viewed.	43
Figure 4-1 Operational tasks and associated components pertaining to ‘Leaf function’.....	49
Figure 4-2 Operational components and associated tasks pertaining to ‘Router’ function’....	50
Figure 4-3 Operational components and associated tasks pertaining to ‘Gateway’ function..	51
Figure 4-4 Example topological re-orchestration from (a) multi-hop network to (b) star network brought about by simple network manipulation.	53
Figure 4-5 Example topological orientations an ‘IoT’ wireless sensor network can adaptively re-orchestrate to as a result of ‘software-defined’ re-orchestration: (a) Multi-hop ‘Topology’ and (b) Star Topology [26].	54
Figure 4-6 Example unified firmware consisting of operational components pertaining to both ‘Leaf’ and ‘Router’ functions.	57
Figure 4-7 Generic firmware block diagram towards allowing for inculcation of additional functions with time.	58
Figure 4-8 Proposed IoT-based sensor network organization based on the Industry 4.0 ideology [25-26].....	60
Figure 4-9 Generic schematic of a Typical WSN.....	64

Figure 5-1 Pseudo code for the (Contiki-based) firmware used for configuring TI CC2538 with the WSN Leaf function.	70
Figure 5-2 Pseudo code for the (Contiki-based) firmware used for configuring TI CC2538 with the WSN Router function.....	73
Figure 5-3 Gateway unit realized using TI CC2538 Evaluation module (EM) in conjunction with the Raspberry Pi 3 (Model B V1.2) device.....	74
Figure 5-4 Pseudo code for the (Contiki-based) firmware used for configuring TI CC2538 with the WSN Gateway function.	75
Figure 5-5 Python script within Raspberry Pi using the ‘REST API’ to read serial data from TI CC2538 and escalate data to the remote ‘cloud server’.....	77
Figure 5-6 Example pseudo code for a (Contiki-based) unified firmware.	82
Figure 5-7 Dynamic re-orchestration of sampling rate through Contiki-provisioned software control to sample data at an increased rate for improved accuracy of (critical) data captured - (a) Lower data communication rate and (b) Higher data communication rate.	84
Figure 5-8 Virtual 2-node point-to-point network depicting the Cooja nodes 1 and 2 configured as the ‘Leaf function’ and ‘Gateway function’ nodes respectively.	85
Figure 5-9 Impact of service rate-based node-level re-orchestrations on total incoming data packets serviced.	87
Figure 5-10 Impact of service rate-based node-level re-orchestrations on total incoming data packets lost.....	87
Figure 5-11 Impact of service rate-based node-level re-orchestrations on the gateway node buffer filled.	88
Figure 5-12 9-node virtual network in running condition within the Cooja simulator.....	92
Figure 5-13 Graphical comparison of CSMA and TDMA network protocol instances with respect to packet loss for the virtual 9-node star network implemented within Cooja.....	92
Figure 5-14 Schematic representing the various core and multi-functional capabilities that can be assumed by the TI CC2538 device owing to Contiki-based software control.....	94
Figure 5-15 Physical implementation of a 9-node TI CC2538-based sensor network within our laboratory premises.	96
Figure 5-16 Graphical trend of ‘sensor’ data (ambient light, temperature and RSSI) captured by the 8 leaf nodes, retrieved from server database.....	96
Figure 5-17 Overview of the setup devised towards facilitating for incorporation the of element of reality within the cloud-based virtualization platform in ‘real time’.....	97
Figure 5-18 Block diagram depicting Remote server implementation and the inter-relation among its various components.....	98
Figure 5-19 (a) Virtual three-node multi-hop network implemented within Cooja; (b) ‘Mote output’ window within Cooja reflecting the dataflow within the multi-hop network.	100
Figure 5-20 (a) Star topology post re-orchestration of the ‘multi-hop’ network (depicted in Fig. 4); (b) Star topology behavior of the re-orchestrated network depicted by the ‘Mote output’ window within Cooja.....	100

Figure 5-21 Certain ‘topological’ orientations that a given IoT-enabled sensor network could flexibly re-orchestrate to as a result of ‘software-defined’ re-orchestration:- (a) Multi-hop ‘Topology’ and (b) Star ‘Topology’.	101
Figure 5-22 Communication messages exchanged amongst the constituent network elements to fulfil the desired re-orchestration process of ‘function swap’ process between the leaf and router nodes.	103
Figure 5-23 Initial state of the three-node network wherein nodes 1, 2 and 3 are pre-configured to behave as leaf node, router node and Gateway node respectively; b) Final state of the three-node network wherein the network has undergone re-orchestration owing to swapping of functions amongst node 1 (now a router node) and node ‘2’ (now a ‘leaf’ node) causing the dataflow within the network to get altered.	104
Figure 5-24 Overall latency (associated with the ‘function-swapping’-based re-orchestration process) deduced through summation of time intervals the individual message ‘ticks’.	104
Figure 5-25 Cooja-based virtual representation of a 6-node network facing impending network fragmentation owing to departure of a mobile router node away from range of connectivity.	105
Figure 5-26 Sequence diagram showing the messages transpiring over the three re-orchestration phases in pursuit of electing the most suitable leaf node to take up the role of the replacement router.	107
Figure 5-27 Messages related to the outcome of the election process as seen within Cooja’s ‘Mote output’ window.	108
Figure 5-28 Screenshot of Cooja’s ‘Mote Output’ showing the time instant of node 1’s functional transformation from the role of a ‘leaf’ node to that of a ‘router’ node and subsequent data acquisition from its constituent leaf nodes.	109
Figure 5-29 Physical implementation of the multi-hop network (corresponding to figure 9) established within the SeNSe lab.	110
Figure 5-30 Data retrieved from SeNSe Lab’s server depicting degradation of the RSSI signal of the departing router (node 5) as it moves away from the Gateway node.	111
Figure 5-31 Cooja nodes configured within the Cooja simulator with the real values obtained from physical experimentation.	112
Figure 6-1 A schematic representing a cloud-based WSN organization for forest fire monitoring purposes.	115
Figure 6-2 Example schematic of WSN deployed for forest fire monitoring purposes wherein one a certain primary level router may is suffering from low battery voltage and is about to die (requiring the network to undergo re-orchestration).	116
Figure 6-3 Screenshot of the virtual network showing only the area of the overall WSN consisting of the cluster wherein member leaf nodes are affected by the ‘dying’ router node.	118
Figure 6-4 Sequence diagram depicting the various messages exchanged amongst the various nodes partaking in the election process.	120
Figure 6-5 Virtual network implementation wherein the ‘dying’ router broadcasts message ‘MRD’ to all the relevant nodes.	122

Figure 6-6 Static model implemented within the virtual Network: Number of participant reachable leaf child nodes scaled up to six nodes.....123

Figure 6-7 Cooja simulator outcome confirming consistence of the result.123

List of Tables

Table 2-1 Distinctions between Modelling, Virtualization and NDT.....	15
Table 3-1 ‘Comparison’ table of operating systems used in WSN.....	29
Table 3-2 ‘Current consumption’ values of TI CC2538 for the two sleep modes.....	32
Table 3-3 Real RSSI values recorded for the departing router (with respect to the stationary leaf node) via physical experimentation.	34
Table 5-1 Table showing the transmission powers that can be selected to configure the TI CC2538 wireless transceiver and their corresponding hexadecimal values to be used within relevant section of the code for the same.....	89
Table 5-2 The ‘impact’ of increasing ‘data communication’ rates on the ‘packet loss’ experienced by the network consisting of eight nodes for various scenarios.	102
Table 5-3 Example representation of mobile router communication strength.	106
Table 5-4 Real RSSI values recorded for the departing router (with respect to the gateway) via physical experimentation.	111

List of Abbreviations

ADC	Analog-to-Digital
API	Application-Programming Interface
APTEEN	Adaptive Threshold - Sensitive Energy Efficient Network
AR	Arrival Rate
ARM	Advanced RISC (Reduced Instruction Set Computer) Machine
AUT	Auckland University of Technology
BS	Base Station
CAPEX	Capital Expenditure
COAP	Constrained Application Protocol
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
HTTP	Hyper Text Transfer Protocol
FFD	Fully Functional Device
IDE	Integrated Development Environment
IETF	'Internet'-Engineering Task Force
IoT	Internet-of-Things
IP	Internet-Protocol
IPv6	Internet-Protocol Version-6
IT	Information Technology
I2C	Inter-Integrated Circuit
LEACH	Low-energy adaptive clustering hierarchy
LPP	Lightweight Presentation Protocol
MAC	'Media Access Control' Layer
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
MySQL	My Structured Query Language
NFV	Network Function Virtualization

NOOBS	New Out of Box Software
NV	Network Virtualization
OPEX	Operational Expenditure
OS	Operating System
OT	Operational Technology
PHP	Hypertext Pre-processor
PM1	Power Mode 1
PM2	Power Mode 2
PNF	Physical Network Function
PPS	Packets Per Second
PSC	Physical Sensor Cloud
PSN	Physical Sensor Network
QoS	Quality-of-Service
RAM	Random-Access-Memory
RDC	Radio-Duty Cycle
REST	Representational-State-Transfer
RF	Radio Frequency
ROM	Read Only Memory
RPL	IPv6 Routing-Protocol-for-Low Power and Lossy Networks
RSSI	Received Signal-Strength Indicator
RX	Reception
SD (card)	Secure Digital
SDN	Software-Defined Networking
SDWSN	Software-Defined Wireless Sensor-Network
SeNSe	Sensor Network and Smart Environment Research Centre
SoC	System-on-Chip
SPI	Serial-Peripheral-Interface
SR	Service Rate
TCP	Transmission-Control-Protocol

TDMA	Time-Division Multiple Access
TEEN	Threshold sensitive Energy Efficient sensor Network
TX	Transmission
UART	Universal-Asynchronous Receiver/Transmitter
UDP	User-Datagram Protocol
uIPv6	Micro IPv6 (Internet-Protocol version 6)
VNF	Virtual Network Function
VSC	Virtual Sensor Cloud
VSN	Virtual Sensor Network
Wi-Fi	Wireless Fidelity
WSN	Wireless-Sensor-Networks
6LoWPAN	IPv6-over Low Power Wireless-Personal-Area-Networks

Chapter 1

Research Motivation, Direction and Thesis Organization

1.1 Introduction

This chapter commences with a generic overview of wireless sensor networks and the typical ‘underlying’ structural approach adopted whilst ensuing upon practical implementations for the same. It then delves into the analysis of certain such sensor network implementations, mainly from a topological standpoint, besides highlighting some of the variances at the functional level. Based on this analysis, the core functional components that constitute any sensor network organization are identified and put forth. The motivation behind driving towards formulation of a dynamically flexible WSN that could have the ability to re-orchestrate its operational behavior at the node and network levels, so as to cope with dynamic service and re-orchestration demands, has been emphasized by means of an example application of forest fire monitoring. Certain factors associated with conventional WSN architectures which tend to hinder their flexible operation are stated before laying the research question. Subsequently, the aim of the research work along with the relevant objectives to be pursued towards execution of the same are stated. The latter stages of the chapter are devoted towards brief discussions on the objectives so specified towards realization of a Software-Defined Wireless Sensor Network (SDWSN) organization. These include modularization of the core WSN functionalities so identified, virtualization of the functions so modularized, re-usability of the modular functions as a means to pave the way for ‘re-configurability’ approach and finally, maintenance of an evolvable ‘library’ of such re-usable modules towards further augmenting the flexibility that could be offered by the proposed WSN organization. This is followed by the conclusion of the chapter.

1.2 ‘Wireless Sensor Networks’

‘Wireless Sensor Networks’ (WSNs) are typically constituted of power-constrained sensor devices that may be deployed in low, medium or high volume for capturing requisite environmental i.e., physical data from its surroundings[1-10]. Each such independent sensor-transceiver node consists of an in-built wireless (transceiver) communication module or interface, enabling it to communicate and exchange the acquired sensed data with other constituent nodes. The individual sensor nodes constitute a network amongst themselves so as to perform or execute a WSN monitoring or sensing task in a collaborative manner. The logical operation executed by a wireless transceiver depends on the function (or software code) with which it (i.e., the microcontroller within it is) is configured.

Continual advancements in the fields of digital IC and SoC technology have considerably contributed to conceiving more powerful microcontroller chips, capable of wirelessly communicating the sensed radio-data packets over the Internet. This major technological breakthrough has paved the way for incorporation of IoT as a prospective solution for pervasive WSN monitoring[11-15]. This serves as the basis for IoT-based sensor networks, wherein

Gateways, via performing the requisite protocol conversion, could escalate the sensed data to a remotely governing cloud server, wirelessly over the Internet[16-20].

1.3 Typical WSN Implementations (Structural Standpoint)

WSN deployments, in general, tend to adhere to a common structural formation but may differ from a topological standpoint depending on factors such as the application for which they are employed [21] (to a certain extent,), the nature of deployment (dense or sparse).

Large-scale WSN organizations, such as those deployed for forest fire monitoring tend to encompass a multi-layered hierarchical architecture, involving aspects such as clustering (including multiple levels of clusters, especially within tree-based topological orientations), multi-hopping, such as on depicted in figure 1-1.

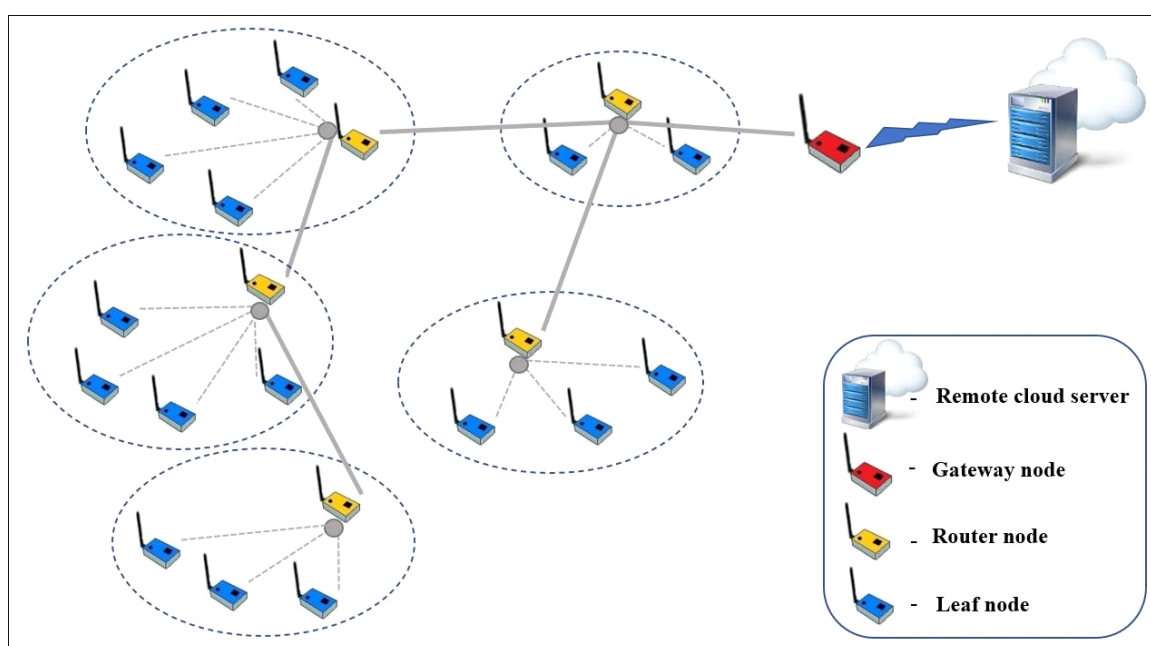


Figure 1-1 Underlying structural approach adopted for practical WSN implementations.

Herein, spatially distributed nodes (denoted in blue) that are equipped with both sensing and communication capabilities capture the physical, real-world data from their surroundings and transmit it over to an upper layer node, in accordance with the protocol employed. These nodes are referred to as ‘leaf’ nodes and are generally grouped (mostly based on geographical vicinity) to form ‘clusters’ within the overall network.

Each such ‘cluster’ is governed by an upper-layer ‘router’ node that tends to act as a cluster-head for the constituent nodes within its cluster and routes the data acquired from them either to the gateway or a higher-level router node.

The network depicted in figure 1 is an example of a ‘multi-hop’ network since sensed data packets emanating from a leaf node traverse through multiple nodes, including multiple levels of ‘routing’ nodes prior to reaching the gateway node.

Nodes configured with the ‘Gateway function’ serve as a ‘sink’ for the incoming ‘sensed’ data relayed by all the routers nodes present within the network. After the requisite protocol conversion, the gateway-node then escalates the sensed data over to the remote cloud server over the Internet. In other words, it serves as an access point to the remote cloud server via acting as a bridge between the leaf sensors and the cloud platform (for both upstream flow of the sensed data and downstream reconfiguration of PSC end devices over the Internet).

A large-scale WSN may consist of more than one node configured as a gateway. The motive behind having in place multiple gateway nodes emanates from multiple network performance-associated aspects. The presence of multiple gateways allows for distributed flow of data over the Internet, thereby significantly mitigating the perennial issues of excessive communication overhead and data congestion and/or collision, detrimentally plaguing the performance of large-scale IoT sensor networks. Besides preventing rapid energy consumption of the gateway-configured nodes, it improves the reliability via eliminating the risk associated with a singular (or small number of) points of failure.

It can be argued that structural formations (wherein the flow of data originating from leaf nodes reach the gateway node via the respective router nodes and eventually the remote cloud server) depicted in figure 1-1 can be generically applied to any WSN implementations for fulfilling any sensor network monitoring or service requirement. Examples of allied structural formations, varying in their topological standpoint that have been adopted for practical WSN implementations are described with their pros and cons below.

Certain precision agriculture-based sensor network implementations, such as the one deployed by [22], adopted a star-topological approach for monitoring and control of environmental parameters. Herein, light, temperature and humidity data captured by the leaf nodes are forwarded to a centrally located intermediate router-node, which in turn, forwards the data to a ‘Gateway’ node, as shown in figure 1-2 [22] below.

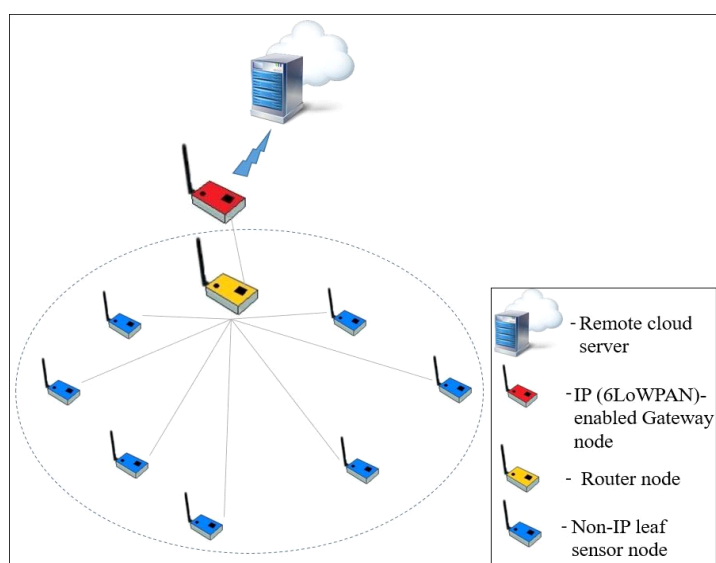


Figure 1-2 Star topology-based WSN implementation for precision agriculture-based application.

As compared to a star topological framework, a tree-topology based network tends to offer increased coverage owing to the presence of multiple routers acting as cluster heads for their respective cluster of leaf nodes. By means of multi-hop communication amongst routers present at the various levels of the tree-structure (with the router closest to the Gateway being the upper or primary level router, second closest to the gateway being the middle or secondary level router, lower level router and so on) [22], data sensed by the leaf nodes are escalated to the gateway node. This topological arrangement, however, suffers from the disadvantage of heightened complexity and greater energy consumption [23]. Such topological implementations are well suited for applications requiring a sizeable area to be monitored e.g., forest fire monitoring. Ammar and Souissi [24] adopt a Zigbee-based tree-topological framework for their real-life testbed to detect forest fire outbreak(s), as shown in figure 1-3 [24].

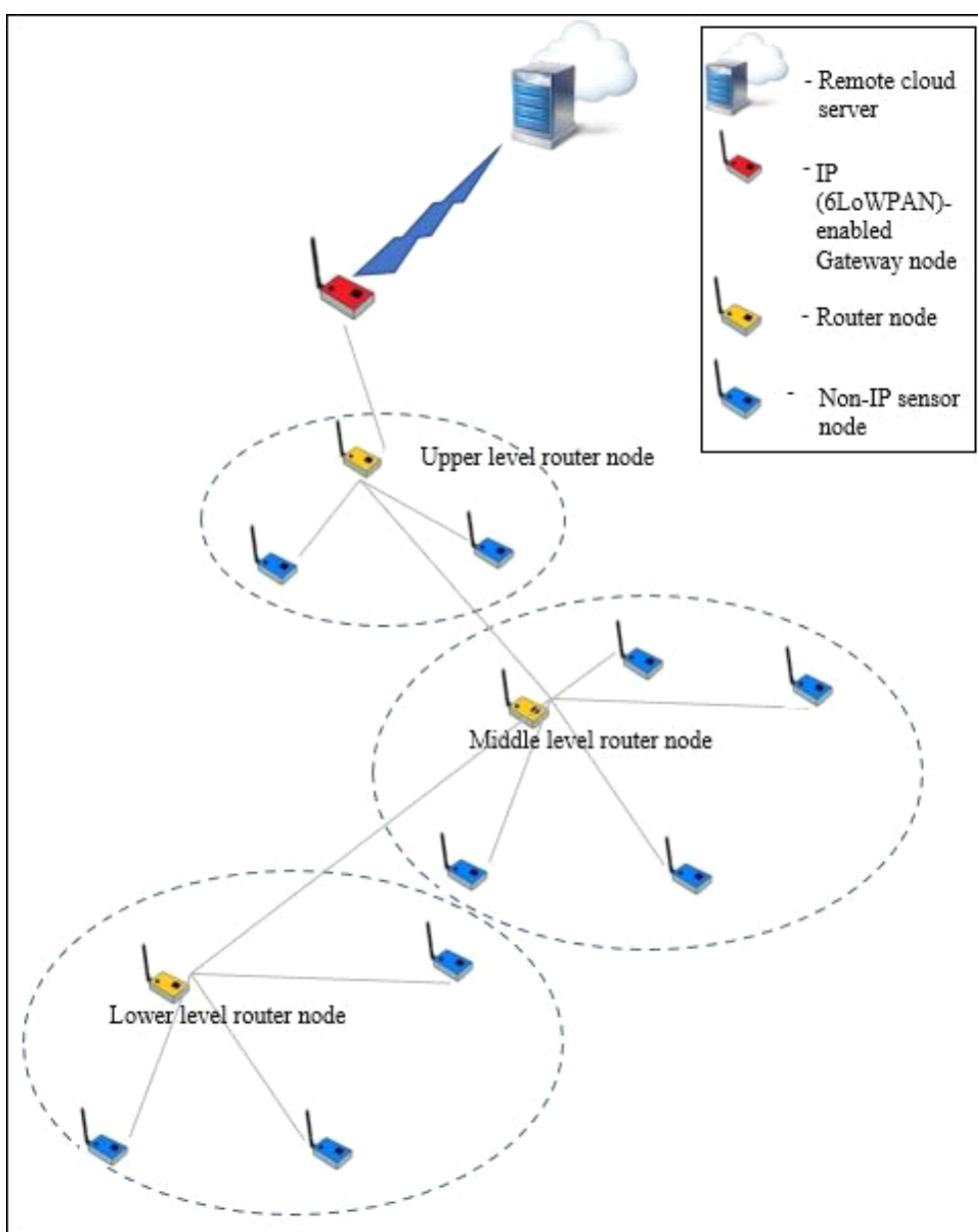


Figure 1-3 Tree topology based WSN implementation for forest fire detection.

It is important to bear in mind that all leaf nodes may or may not be endowed with 6LoWPAN or allied IP capability (depending upon the resources encompassed within their hardware and/or software architecture). Figure 1-4 presents example of mesh-topology based network wherein certain leaf nodes capable of communicating directly with the cloud over the Internet could do so by virtue of the 6LoWPAN protocol (embedded within them). The non-IP based sensor nodes could forward their data to such 6LoWPAN-enabled counterpart leaf nodes so that their sensed data too, could be escalated over to the cloud over the Internet.

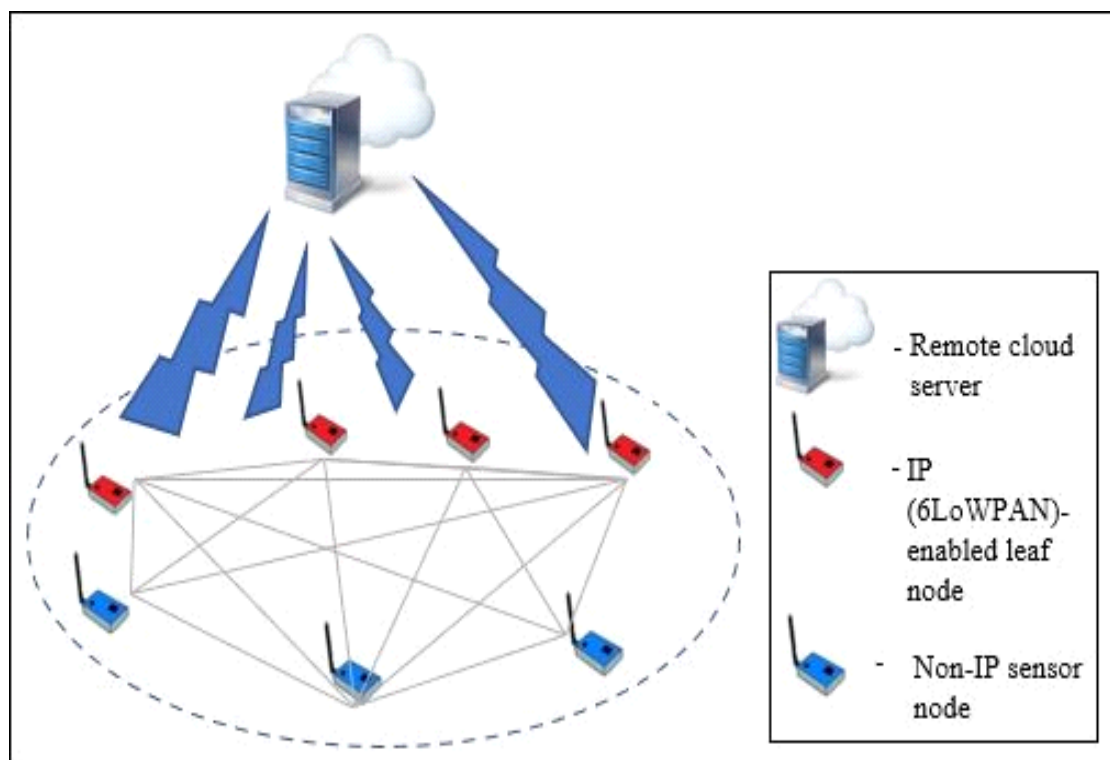


Figure 1-4 An example of a mesh-topology based WSN with certain 6LoWPAN enabled sensor leaf nodes.

Based on the above account, it can be quite discernibly stated that regardless of the topological variations, any WSN organization tends to be composed of the three key fundamental, standalone functional components, namely the Gateway-function, Router-function and Leaf-function [25-26].

1.4 Motivation for Driving Towards a Dynamically Flexible WSN

WSNs are employed for a variety of applications, viz., environmental, industrial, healthcare monitoring [27-36], etc. The nature of service requirements entailed by such applications dictates the nature of WSN deployments with respect to factors such as volume, density, topology, among others, such as channel access method, routing mechanism, sensing and data acquisition requirements. Each of the application-specific service requirements, however,

along the dynamics of the monitored physical phenomenon, may vary dynamically in an unpredictable manner necessitating duly responsive monitoring capabilities on part of the WSN employed. Furthermore, network fragmentation events arising out of events such as node-death caused by low battery power (or say, departure of a mobile node away from the network-connectivity chain) that could adversely impact the flow of data within network or result in loss of data, must be resolved in a seamless manner. WSNs, thus, ought to be able to undergo flexible re-orchestrations so as to adequately cope with the diverse service requirements emanating out of variable dynamics of the monitored phenomenon, as well as network re-organizational requirements, whilst engaging with physical environments.

Consider the example application of forest fire monitoring wherein flexible operation of the large-scale WSN deployment is crucial towards real-time response to a fire outbreak event so as to prevent its uncontrolled spread. Flexible re-orchestration of such implementations from a topological standpoint is also of critical value towards overcoming network fragmentations that may be encountered from time to time. Upon detection of a fire outbreak, the leaf nodes deployed in vicinity must be able to undergo requisite re-orchestration with respect to their node-operational parameters viz., data buffering size, data communication rate, amongst others, to be able to capture vital information during very initial stages. Besides these node-intrinsic adjustments, the entire cluster of leaf nodes within that particular location of the monitored region, along with their respective router-cluster head node, could switch to a favourable channel-access method like sensor-data polling or a multiple access scheme like time division multiple access (TDMA)-based channel access. Such a network-level re-orchestration could further expedite the flow of data to the remote cloud server via the gateway. In the event of network fragmentations caused either due to depletion of battery power of any of the intermediate router nodes or as a result of any of them suffering fire damage, the most suitable node nearby must be able to undergo functional re-orchestration to take up the role as a replacement router. If need be, the entire network as a whole, may need to dynamically adapt to a different topological orientation to maintain network connectivity. The aforementioned possible scenarios that the WSN deployed for forest fire monitoring may be required to adapt to, from time to time, highlight the demand for flexible re-orchestration across multiple levels within the network.

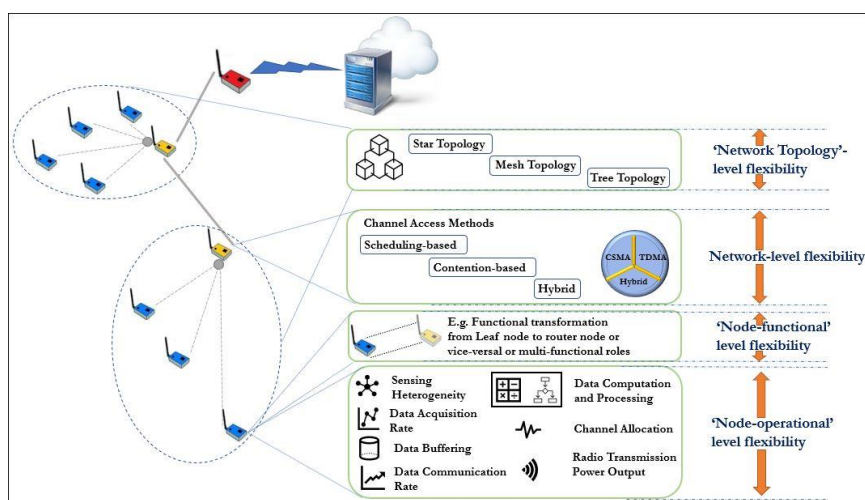


Figure 1-5 Scope for flexible re-orchestration across multiple levels within a WSN.

Depending on the scenario necessitating re-orchestration, a WSN must be able to undergo flexible re-orchestration at the node-operational, node-functional (leaf, router, gateway or functional), network operational levels (e.g., channel access methods) and topological levels (star, mesh and tree topological formations. amongst others), as depicted in figure 1-5.

Node-level re-orchestrations imply manipulation of solely the intrinsic node-operational parameters pertaining to a given WSN function (via software control) which neither implicate in any way on the functional role being executed by it nor at the network or topological levels. For example, node-operational parameters pertaining to. say, a leaf node that could be re-orchestrated via software control include selection of the requisite sensor (provided the node has multiple different sensors at its disposal, acquisition of the sensed data, buffering, computation, allocation of requisite channel, (radio) transmission power and data - communication rate.

Re-orchestration at the node-functional level could be facilitated through reconfiguration (e.g., transformation of leaf node either to a router node or gateway node or vice-versa, or execution of multiple functionalities simultaneously during running condition, provided the hardware is capable of accommodating for and executing them). In certain cases, such node-functional transformation results in the topological re-orchestration of the network including its inherent dataflow. Figure 1-6 [25-26] depicts some of the topological variations of a given 9-node WSN, emanating as a result of software-defined re-orchestrations. Consider figure 1-6a. Node-functional reformulation of node 4 via software control to that of a router node, results in overall network topological re-orchestration (of the IoT-based sensor network) from being a star-network (as shown in figure 6a to a tree-network (as illustrated in figure 1-6 b). In a similar fashion, the IoT based network could (be made to) undergo certain other software-defined re-orchestrations to assume mesh and multi-hop network topologies, as represented by figures 1-6c and 1-6d, (respectively). In all the topological orientations, the three core functions remain an integral (consistent) part of the WSN in consideration.

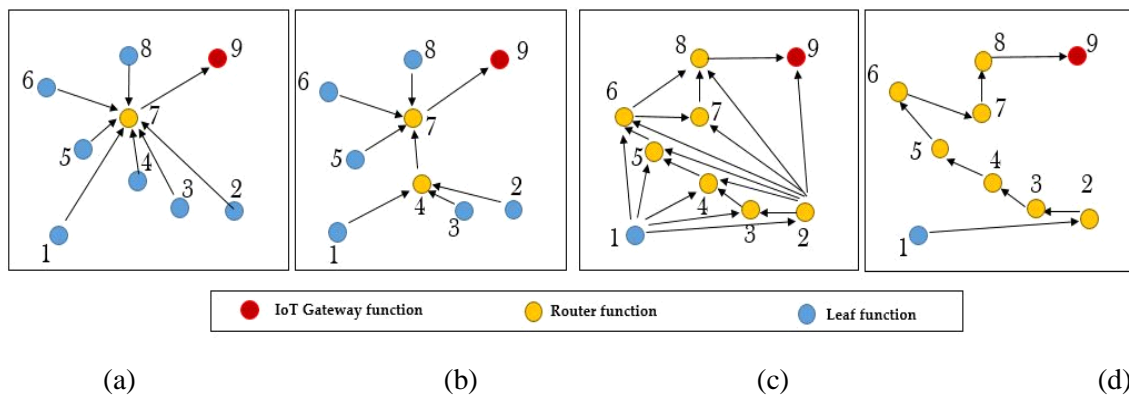


Figure 1-6 Certain example topologies an IoT-based sensor network could be re-orchestrated to, by means of software control: (a) Star-network ‘topology’; (b) Tree-network topology; (c) Mesh-network topology; (d) Multi-hop-network topology.

Network-level re-orchestrations do not implicate in any way on the topological orientation of the network but tend to significantly alter the nature of flow of data taking place within the network. For example, manipulation of the relevant parameters residing within the MAC layer may cause the network to switch its operation in accordance with a different channel access method (e.g., switching to a TDMA-based channel access method from a CSMA-based one).

Traditional WSN architectures tend to suffer from a plethora of deficiencies ranging from lack of operational flexibility, complex maintenance and management, time-consuming implementation processes, amongst certain other factors, to no scope for remote configurability owing to lack of centralized control [25]. Besides these, time and labour costs associated with manual reconfiguration, along with network downtime arising out of such disruptions render such a prospect infeasible [37].

With regard to the above, flexible operation of such WSN deployments, backed by the intelligence and softwarization offered by a cloud platform, is of vital importance as it engages with its physical surroundings. Incorporation of the ideology proffered by the emerging softwarization paradigm of SDN tends to be prospective in this regard. Owing to the separation of the data plane from control plane, as well as provisioning for centralized control and intelligence, certain principle advantages offered by the SDN include flexible configuration control, time-efficient implementation and operation and low maintenance and management costs, coupled with remote configurability[26,38-39].

By means of relying upon a software solution, as opposed to hardware solutions through incorporation of an enabling technology such as virtualization, (including provisioning for pertinent soft-trialling of numerous re-orchestration scenarios,) an SDWSN could lend itself as a viable solution towards seamlessly resolving dynamic re-orchestration and service demands in real-time.

1.5 Research Questions, Aims & Objectives

The above account leads to the following question that has acted as the driving force for this particular research work:

What would be the architectural organization for a typical large-scale WSN that offer flexible re-orchestration(s) and able to react to dynamic service demands?

The ‘aim’ of this ‘research’ work, therefore is ‘*To converge upon a cloud-based software-defined sensor network organization*’.

The (research) objectives towards executing the aforementioned aim are as follows:

- *To identify and define the generic WSN functional components so as to render them as software modules that could constitute a firmware (either individually or as a combination of multiple such software modules).*
- *To virtualize the software modules so defined.*
- *To identify, justify and implement the approach (or method) that is probably the best suited for realizing flexible WSN re-orchestration at both physical and virtual levels in a responsive manner.*
- *To establish a cyber-physical architectural organization that could facilitate for the necessary soft trials of the virtualized software modules so as to converge upon and implement the most suitable re-orchestration onto the physical WSN*

1.6 Thesis Organization

The thesis comprises of a total of seven chapters, each offering information pertaining to a specific aspect of the research work. Chapter 1 lays the necessary background, vision, motivation and direction behind the research work undertaken. A review of the related literature pertaining to the important elements associated with this research viz., WSN softwarization, virtualization, re-orchestration (including the downtime associated with the same) as well as similar other ‘architectural’ frameworks (designed with relatively similar objectives in mind) have been documented within chapter 2. Chapter 3 offers information on the virtualization and hardware tools employed towards concept simulation, (implementation) and testing purposes. Chapter 4 revolves around the development of the proposed ‘Software-Defined Wireless Sensor Network’ (SDWSN) concept. It firstly spells out the pre-requisites for the same prior to delving into the key aspects of the core modular WSN functional components, SDWSN approach adopted and virtualization, among others. The proposed architectural organization is described herein, after which the strategy adopted for the re-orchestration process (including formulation of a generic model for the latency that the network may experience as a result of the same) is laid out. Based on certain example cases of (node and network-level) software-defined re-orchestrations depicted in chapter 4, chapter 5 offers information on aspects pertaining to the physical and virtual implementations of the same. Contiki-oriented pseudo codes of the firmware developed and employed for configuring both the real-life Contiki-ported Texas Instruments CC2538 wireless transceivers as well as the virtual *Cooja* motes within the *Cooja* virtualization platform (offered by Contiki) have been put forth. The intention herein was to put forth the viability of employing Contiki as a firmware-development tool that caters for virtualization as well, thereby enabling convergence of suitable re-orchestrations via conduction of soft-trials (at the virtual level). Chapter 6 puts forth a use case related to forest-fire monitoring in an attempt to demonstrate the applicability of the proposed concept in a real-world setting or situation. Herein, it is elaborated that re-orchestration demands arising out network fragmentation events could be resolved via re-orchestration (in accordance with the re-configurability approach) of the requisite constituent node(s). The conclusion of the research work, along with a section devoted to the future work, is documented in chapter 7.

1.7 Research Outcomes - Publications Based on the Thesis Work

A total of six papers have been published over the course of this PhD research work. Five of these have been published within conference proceedings of reputed international publishers such as IEEE, ACM, SciTePress while one of them was published within the prestigious IEEE ‘Internet of Things’ journal. The conceptual basis behind the proposition advanced within this research work (that of driving towards a cloud-based software-defined sensor network organization as well as aspects pertaining to its implementation and performance) has been reflected within these publications. The conference publication presented remotely within the online streaming-based event of the 9th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS 2020) was adjudged to be the ‘best student paper’.

The list is provided below:

Journal Publication:

1. Ezdiani, S., Acharyya I., Sivakumar, S., and Al-Anbuky, A., "Wireless Sensor Network Softwarization: Towards WSN Adaptive QoS" IEEE INTERNET OF THINGS JOURNAL, VOL. 4, NO. 5, OCTOBER 2017, PP. 1517-1527.

List of Conference Publications:

1. Acharyya, I. and Al-Anbuky, A., "Software-Defined Wireless Sensor Networks: WSN Virtualization and Network Re-orchestration," 9th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS), Online Streaming, 2-4 May 2020 (*Received the 'Best Student Paper' Award at the conference.*)
 2. Acharyya, I., Al-Anbuky, A. and Sivakumar, S., "Software-Defined Sensor Networks: Towards Flexible Architecture Supported by Virtualization," 2019 Global Internet of Things Summit (GIoTS), 17-21 June 2019, Aarhus, Denmark.
 3. I. S. Acharyya and A. Al-Anbuky, "Towards Wireless Sensor Network Softwarization" 2016 IEEE International Workshop on SDN and IoT, June 6, 2016, Seoul, South Korea.
 4. S. Ezdiani, I. S. Acharyya, S. Sivakumar and A. Al-Anbuky, "An IoT Environment for WSN Adaptive QoS," 2015 IEEE International Conference on Data Science and Data Intensive Systems, Sydney, NSW, 2015, pp. 586-593.
 5. Syarifah Ezdiani, Indrajit S. Acharyya, Sivaramakrishnan Sivakumar, and Adnan Al-Anbuky. 2015. An Architectural Concept for Sensor Cloud QoSaaS Testbed. In Proceedings of the 6th ACM Workshop on Real World Wireless Sensor Networks (RealWSN '15). ACM, New York, NY, USA, 15-18.
-

1.8 Conclusion

Flexibility is critical to any WSN organization's utility as an autonomous monitoring resource. WSNs ought to be able to satisfactorily adapt to the dynamic service requirements whilst engaging with a given monitored phenomenon necessitating flexible operability at both node and network levels, including that from a topological standpoint. Development of a cloud-based software-defined sensor network organization that allows for virtualization and soft-trialling of numerous re-orchestration scenarios has been deemed as a viable solution in this regard. Besides obviating the requirement of the underlying physical network for testing purposes (and thereby eliminating any disruption to the ongoing sensing and data acquisition process ensuing within the physical layer), such a cloud-supported WSN organization could also offer information pertaining to the implications of imposition of a certain re-orchestration outcome on the physical network, prior to the actual implementation. Prior to realization of the proposed organization, the three functionalities integral to any WSN organization are identified through relevant literature analysis of structural approach, adopted for typical WSN implementation. The key pre-requisites entailed by the aforementioned proposition viz.,

modularization, virtualization, approach adopted for flexible re-orchestration i.e., Re-configurability and maintenance of a library of software modules (for augmenting the flexibility of the given WSN) are laid out.

Chapter 2

Literature Review

2.1 Introduction

Lack of software control, coupled with ad-hoc mode of operation may exacerbate non-adaptive operational dysfunctionalities leading to significant performance degradation in conventional Wireless Sensor Networks (WSNs), especially when operating within a dynamic IoT environment. Flexibility of complex IoT-enabled sensor network infrastructures is an important requirement as it engages with its physical surroundings [25, 39-40]. As discussed in chapter 1, WSNs deployed for dynamic outdoor applications such as forest fire monitoring ought to not only detect and swiftly react to fire outbreaks but also adaptively resolve network fragmentation events that it may encounter from time to time. In order to adequately comply with such demanding service requirements, WSN organizations ought to be able to flexibly re-orchestrate their operational behaviour both at the node and network-operational levels, including their topological orientation. As a means to achieve this, it is imperative to exert software control over the operational parameters available at the node and network levels. To a certain extent, IoT-based cloud computing platforms tend to act as a viable solution in this regard since it encompasses a multitude of software-based configuration capabilities, including virtualization, data storage, data analytics, etc., which can potentially facilitate optimal remote configuration of the low-power nodes over the Internet [25-26, 41-51]. A cloud-based solution, inclusive of virtualization and requisite software control can significantly lean out the WSN network configuration process, thereby making a worthy contribution towards the flexibility and swiftness of the network in reacting and capturing random physical events [41, 52-55].

This chapter offers an account of the existing state-of-the-art pertaining to related aspects like virtualization, softwarization, and re-orchestration (as applied to Cloud-based architectural frameworks) that a typical software-defined sensor network organization may tend to entail. Specific focus is laid on the aspects of flexibility proliferation and latency reduction brought about via incorporation of such cloud-facilitated technological assets, whilst attempting to scrutinize the extent of their effectiveness. Based on the analysis, the novelty of the proposed paradigm in advancing the state-of-the-art is highlighted including identification of certain key constituent elements/conceptual pre-requisites.

2.2 WSN Virtualization

The term Virtualization implies abstraction of the underlying physical functionalities into logical or virtual functionalities, allowing them to be utilized in an efficient manner [56-57]. This technological asset tends to offload numerous significant benefits when incorporated or applied within both wired and wireless networking environments viz., increased flexibility, running multiple applications at the same time on the same infrastructure [57-59], manageability, etc. [59]. Within the context of WSN, by virtue of allowing for exact replication of the logical facet of the underlying physical function, it opens the door for conduction of

near-accurate soft-trials, thereby virtually completely obviating physical hardware nodes for testing purposes. This serves towards easing out of the complexities associated with conduction of testing or trials involving real-life networks besides reducing the service and/or operational costs, time, and labour associated with WSN reconfiguration, in general to a significant extent. Khan et. al [57-58] duly highlight the need for virtualization of WSNs deployed for real-world applications such as forest fire monitoring, citing the aforesaid reason pertaining to enabling multiple users to access the common ‘resource’ or ‘infrastructure’ in a simultaneous fashion. Certain state-of-the art tools available nowadays such as Contiki’s Java-based Cooja network simulator can be utilized as virtualization platforms, albeit for selected target hardware (notes e.g., Texas Instruments’ CC2538 Evaluation Module). Broadly speaking, WSN virtualization could be classified into the two different categories of ‘Node’ and ‘network-level’ virtualization [59-60].

2.2.1 ‘Node-Level’ WSN Virtualization

Node-level virtualization involves enablement of multiple applications, operating in a virtualized environment, in isolation from the physical node [60]. Herein, the middleware provisions for creation of such multiple software instances at the virtual level based on the capabilities possessed by the physical node [60]. This has parallels with the concept of Sensor Function Virtualization and has been looked at to avail the desirable features pertaining to multi-serviceability, flexible operability and concurrency. However, the work is not supported by any virtual or physical implementation.

Akram and Gokhale [61] highlight a host of prospective advantages that can be achieved through virtualization of the sensor functions. These include concurrent multitasking capability, scalability, elasticity, elimination of hardware requirement, and most importantly, flexibility. The authors opine that since conception of ‘Sensor Function Virtualization’ (SFV) within a virtualization environment allows for flexible trialling of the various software instances of the virtual functions, the process of execution a variety of sensing operations, as well as sensing service alteration, becomes quite straightforward.

In a bid to enhance distributed processing and intelligence capabilities of network systems operating in IoT environments, Van den Abeele et. al [62] harness the concept of SFV. The authors advocate enhancement of distributed processing by means of transferring functionality of the low power nodes to either a single virtual entity, such a virtual gateway, or to the access network. A shift of functionality to the unconstrained domain of cloud-virtualization results in the advantages of scalability and elasticity. The concept is harnessed by the authors within their proposed flexible architecture wherein on-the-fly deployable, modularized sensor virtual function packages enable real time re configurability. SFV has also been adopted by Van den Abeele et. al [62] to alleviate deficiencies such as lack of edge-processing capability and low scalability affecting DTLS i.e., Datagram Transport Layer Security within their ‘IETF-IoT’ stack-based WSN. The authors, however, do acknowledge that incorporation of SFV for the real-life physical nodes may result in limited flexibility.

Within their multi-tier architectural system, Li et. al. [63] employ the concept of SFV to virtualize the physical level so as to project a single physical component as being capable of offering a multitude of services.

2.2.2 ‘Network-Level’ WSN-Virtualization

While node-level virtualization allows for several applications running their tasks concurrently on a given node [64] (based on the resources encompassed by it) [65], network-level virtualization allows for multiple VSNs (Virtual Sensor Networks) to run on the underlying network, each VSN running a dedicated application at a time [66]. Network-level virtualization may be hosted within the cloud, lending itself as a viable avenue for planning and testing of various (types of network-wide) WSN re-orchestration scenarios such as MAC-layer-based manipulations [60] or topological manipulations [60, 67] (which tend to considerably impact the flow of data within the entire network, as a whole) [25,41-42]. Cloud based virtualization has been adopted in many works and research-based projects. Our research publications (Acharyya, I. et.al, Acharyya, I. and Al-Anbuky, A. and Ezdiani, S. et al.) [25, 41, 50] leverage upon this advantageous feature in the pursuit of edging towards a flexible and responsive sensor network system. Within their multitier cloud architecture, Leon-Garcia and Tizghadam [68] highlight the role of cloud-based virtualization in realization of scalable and flexible applications. Moreover, such virtualization framework also accounts for heterogeneity within their architecture. Besides entirely obviating the requirement of physical sensor nodes to run tests on, such network virtualization technology (when operated in conjunction with other softwarization-based resources present within the cloud) could also play a key role in exploring the degrees of freedom or modes of operation available for network re-orchestration. The ability offered by virtualization environments towards foreseeing the implications of each re-orchestration scenario ‘soft-trialled’ within it (i.e., data lost, or downtime experienced by the network as a result of the same) allows to (relatively) quickly converge upon the most suitable re-orchestration [25-26, 69-70] to be applied onto the underlying WSN.

2.3 Viability of WSN ‘Virtualization’

The advantage offered by virtualization as a technique or avenue for testing network re-orchestration scenarios (in a more accurate manner) to be applied on to physical networks over conventional modelling and simulation tools can be assessed from Table 2-1 below. Virtualization however has certain shortcomings viz., inability to account for real world dynamics, physical factors, etc., besides not necessarily mimicking the processes occurring within the underlying network in real-time. Such deficiencies can only be overcome by employing more advanced technologies such as ‘Digital Twin’.

On comparing the technological assets of virtualization and digital twin, it was noted that while simulation conducted on a virtual platform accurately mirrors i.e., mimics the logical components of a physical object, it may not necessarily take into account the real-world factors affecting the physical network sensing and monitoring (since it is not feasible to model or simulate real-world conditions). This tends to introduce significant deviations within the

simulations so conducted and render it unable to represent prevailing real-world operational characteristics of the device in real time.

Table 2-1 outlines certain key demarcating aspects in regard to the technologies of modelling, virtualization and network digital Twin.

Table 2-1 Distinctions between Modelling, Virtualization and NDT

Conventional Modelling	Virtualization	Network Digital Twin
<ul style="list-style-type: none"> No exact replication of the logical functionality of the underlying physical network leading to a somewhat rough estimation of the network behaviour Significant aspects may not be represented leading to inaccuracies 	<ul style="list-style-type: none"> Exact (one-to-one) logical replication of the underlying network resulting in precise mapping (or mimicking) of network behaviour from the logical standpoint 	<ul style="list-style-type: none"> Exact twin of the underlying network resulting in precise mapping (or mimicking) of network behaviour from both logical and actual (physical) behavioural standpoints Follows up with the process (continuously ensues upon capturing the system process behaviour during run-time)
<ul style="list-style-type: none"> Typically network behaviour is not mapped in real-time 	<ul style="list-style-type: none"> Typically network behaviour is not mapped in real-time 	<ul style="list-style-type: none"> Reflects the real-time behaviour [71] of the underlying system in real-time.
Typically, does not account for the <ul style="list-style-type: none"> real world data real-world dynamics physical factors influencing the behaviour of the underlying physical network. 	Typically, does not account for the <ul style="list-style-type: none"> real world data real-world dynamics physical factors influencing the behaviour of the underlying physical network 	Typically, does account for the <ul style="list-style-type: none"> real world data real-world dynamics physical factors influencing the behaviour of the underlying physical network
<ul style="list-style-type: none"> Incapable of predicting the network behaviour accurately 	<ul style="list-style-type: none"> Capable of predicting the behaviour of the network accurately from a logical standpoint 	<ul style="list-style-type: none"> Capable of forecasting system or network behaviour near-accurately from both logical and actual (physical) behavioural standpoints

While simulation results obtained via conventional modelling tools cannot be said to accurate, Network Digital Twin (NDT) is quite an advanced technology entailing considerable

costs, time and much involved and/or complex tools, debilitating its feasibility for the proposed solution at this stage. In contrast these to technologies, availability of an open-source tool (such as Contiki's Cooja) allowing for seamless logical virtualization has been leveraged upon to converge upon a cloud-based software-defined WSN organization. Through obviation of hardware requirement for trialling re-orchestrations to be applied to the physical world, network virtualization serves as a cost-effective and time-efficient platform to soft-trial network scenarios and observe the implications of a variety of software-defined alterations within the virtual environment as well as explore the degree of freedom procurable from each of the functionalities.

2.4 WSN Softwarization

The paradigm of 'Softwarization' tends to encourage the aspect of virtualization (which involves running a particular functionality in software rather than hardware) and thereby a test environment that allows for soft-trialling or soft-re-orchestrations. The core aspect herein involves decoupling of functions from the physical layer and creation of abstracted or logical instances of the same. Needless to say, there exists plenty of scope for virtualization technology in this regard. By means of 'virtualization' of the physical functionalities, flexible software-based re-orchestrations could be ensued upon in a rather seamless way, considerably reducing OPEX and CAPEX [72]. The two technologies of SDN and NFV that enable softwarization have been employed for numerous WSN implementations [73-76]

2.4.1 SDWSN Node-Level Re-orchestrations

A plethora of works have leveraged on the technological paradigms of network virtualization, SDN, SDR, etc. to render WSNs dynamically reconfigurable [61, 77-78]. Realization of software-defined re-orchestration at an individual node level (within SDWSNs) typically tends to involve (a) decoupling of control plane from the data plane, (b) a centrally located 'SDN controller' (that may consist of a certain number of programmable controller nodes) directing the operation of SDN switches, (c) rendering it open to flexible management via requisite (programming) interface and (d) rendering the individual nodes 're-programmable' via 'wireless' communication [79]. Architectures put forth in this regard such as the ones proposed in [61, 80, 81] also take into account features such as network topology-discovery, memory, data transmission, data acquisition, etc.

The SDN-oriented solution of SDNSensor proposed by Akram and Gokhale [61] is directed towards optimizing the process of 'cross-layer' programmability via employing 'flow-tables'. However, the study lays emphasis only on MAC and Network layers to demonstrate proof of concept. Zeng et. al and Miyazaki et. al put forth a cloud-based architecture wherein a central server presides over the task of generating customized programs for each of the constituent software-defined nodes as well as re-programming them via wireless communication with the same [82, 83]. Generation of the customized programs takes place in accordance with the specifications contained within the 'Scenario Description' unit whilst banking on a library of certain basic functionalities. The aspect of virtualization, however, for

soft-trialling of network re-orchestrations at the cloud prior to implementation has not been considered herein. Moreover, both the works do not specify the strategy adopted whilst going about re-orchestrating the behaviour of the physical nodes (and the network as a whole, including that from a topological perspective) nor focus on the latency or downtime experienced by the network (during re-orchestration).

2.4.2 SDWSN Topological Re-orchestration

In essence, the ideology of SDN revolves around separation of the control and data planes to allow for flexible network management from a centrally located SDN controller [84-87]. Such an approach involving abstraction or logical decoupling of the functionalities from the underlying physical hardware devices has been adopted for a good number of WSN-based research works as a means to solve the problem of dynamic network re-orchestration. In their pursuit of exercising control over the topological orientation of their network, Haque et. al [88] ensue upon devising a logic-based controller (that is centrally located within their *SDSense* network architecture) which not only hosts the requisite routing, scheduling and other control modules but also allows for user-defined assembly and disassembly of software modules.

Jemal et. al [89] target dynamic network re-orchestration through a rather conventional approach involving inclusion of middleware components as a means to adaptively manipulate the operational behaviour of clustered WSNs. Within their approach, the aspect of planning of the desired re-orchestrations is dictated by means of a pre-defined ruleset. A relatively simplistic case of network fragmentation resulting from mobile sensor(s) has been included within their work wherein network re-orchestration occurs as a result of the pertinent sensor node(s) undergoing relevant adaptations and connecting with the gateway once within its communication range. The aspect of network re-orchestration has been almost wholly limited to clustered WSNs. Re-orchestration from the viewpoint of network topology has, strictly speaking, not been considered within this paper. Neither performance evaluation results indicating any improvement in the network's performance upon undergoing software-defined re-orchestrations nor analysis pertaining to the downtime experienced by the network as a result of the same have been documented within the paper. Moreover, the potential benefits of aggregating functional (both core and non-core) as well as requisite knowledge components within a common readily accessible repertoire within the cloud layer, as a means to enhance network flexibility, have not been considered.

Kipongo et. al [80] ensue upon formulation of a software-defined architecture for WSNs wherein the SDN controller is equipped with the capability of both visualizing and managing the topological orientation of the network. The proposition is not supported by means of any evaluation results. The need to develop a protocol for *topology discovery* (whilst keeping the associated latency to a minimum) has been duly acknowledged, followed by a review of the prevailing state-of-the-art for the same. Galluccio et. al [90] propose SDN-WISE, an SDN-centric solution for topological management of WSNs wherein a particular layer within the controller responsible for managing network topology is equipped with virtualization (of underlying physical functions) and (the layered-)stack management capabilities, besides the ability extracting crucial information of the physical nodes such as their battery power levels, radio signal strength, battery capacity, RSSI (Received Signal Strength Indication), address, etc. from the 'underlying' devices and 'relay' them over to the controller(s) and c) exert control over the *stack* layers denoted by the controller(s). In furtherance to this, Abdolmaleki [81] incorporates a fuzzy logic-based network topology discovery protocol that reportedly leads to

prolonging of the lifetime of the WSN as well as packet loss reduction. In their efforts to address the issue of managing node and network-operational aspects of IoT-based sensor networks, Bera et. al [91] design an SDN controller inclusive of both network and node-specific rule-based control policies (an approach which enables them to manage the format of their respective packets). Theodorou and Mamatas [92] develop *Coral-SDN* architecture within which the management of the flow of data within the network is presided over by CORAL centrally located controller. Comprising of a modular subsystem (that hosts certain ‘algorithms’ and ‘rules’) meant for Decision Making, this controller allows for rule-based network routing and topological adaptations. It also consists of a module devoted for ‘network modelling’ purposes. This particular module retains an abstracted, graph-based view of the physical network and hosts RSSI and Link Quality Estimation-based information.

2.4.3 SDWSN Architectural Frameworks

Ndiaye et. al [39] vie to achieve greater flexibility and efficient management of WSNs through formulation of SDNMM, an SDN-based management framework that provides an avenue for swift testing and deployment of modular management entities. The proposed modular management framework is supported by a controller hosted context data knowledgebase, working in conjunction with Management Service Interface (MSI).

The proposed architectural framework is divided into three SDN-based abstraction planes, namely the ‘Application’ plane, ‘Control’ plane and the ‘Data’ plane. The Application plane is dedicated for end-user applications, monitoring of network-wide energy levels and network status with respect to fault detection. Desired configuration of nodes and issuance of context-based network policies are actioned through this plane. The Control Plane is subdivided into two parts. WSN-centric management modules (such as those pertaining to network topology, QoS, energy consumption and security) reside within the upper Global controller cluster, whereas the cluster manager units responsible for execution of tasks (in their respective clusters) are hosted by the ‘Local controller’ part of the control plane. On the whole, the Control plane is accountable for provisioning of APIs to both the application and data planes and administering of policies and flow commands (based on context) to serve application requests. The Data plane is composed of the clustered, physical, SDN-enabled sensor nodes (i.e., a combination of SDN-enabled switch and sensing end device). Sensed data packets (generated by the SDN-enabled node) are relayed over to the sink nodes, which in turn allows for data-linking with nearby cluster manger units. Inter-planar relaying of sensor information, network/context information, etc., is provisioned by means of APIs. The element of Management Service Interface (MSI) employed for this work has been considerably emphasized, along with the aspects pertaining to contexts. Simulations have been performed to evaluate and justify the proposed SDNMM framework.

Herein, although the role and mechanism (or working) of the Management Service Interface (MSI) has been elaborated, the aspect of modularity (in terms of having in place distinct functional components as well as communication APIs) and that the various such modular management entities pertaining to the key management areas of WSN viz., network topology, QoS, security, energy, etc., are stored within the ‘Global Controller’ layer of the control plane have been well emphasized (and realized), there lacks a clear definition of the basic functional roles of the core network components and the scope of additional functionalities which can be

dynamically undertaken/assumed by them, keeping the mind the operational limitations of such low-power transceivers.

The aspect of testability of the ‘to-be-deployed’ or ‘implemented’ modular management entities (or network policies), prior to deployment by means of a virtualized setup has mostly been underplayed. The ‘context data knowledge-base’ has been mentioned but not depicted.

Inclusion of a pseudo code pertaining to the controller and SDN-enabled nodes employed for conducting the Cooja simulation could have offered more insight as to how the simulation and thus, the proposed framework delivers. Moreover, although the monitoring management section details the ‘context data’ approach as well as the program structure, a satisfactory explanation as to how a singular node can undertake responsibilities of MSI (Management Services Interface), in addition to handling the understandable or feasible ‘context data’ aspect in real-life has not been provided. There is no specificity with regards to which management entity was invoked (and which exact parameters with such broad network-management aspects or ‘domains’ were tweaked) for each of the three simulations undertaken, apart from the context base-related aspects. Moreover, how network topology can be influenced through SDNMM framework is unclear (since no results pertaining to this aspect have been documented).

Jemal et. al [89] pursue a similar objective of achieving self-adaptation within WSN by means of adopting a somewhat similar approach. While certain key requisites towards devising an adaptive architecture, viz., adaptive middleware, control elements, simulation aspect, etc. are taken into account, the authors do not lay emphasis on the aspect of adaptive network re-orchestration from a topological point of view. No results pertaining to any improvement obtained as a result of software-defined topological re-orchestration have been presented within the paper. The authors state that the aspect of ‘planning’ of any network adaptation takes place on the basis of ‘pre-defined’ set of rules. The authors consider a case of network fragmentation caused by the event of a mobile sensor node drifting away from its original location. As a means to resolve this ‘network fragmentation’ scenario, the authors enable the node to adapt and connect to the gateway node within its communication range. Investigation pertaining to the ‘network downtime’ aspect is not undertaken.

In contrast to this, the router cluster-head election or replacement-based example case considered (and explained) in chapter 5 seeks to demonstrate the re-orchestration process (in its entirety) as well as offer information pertaining to the downtime incurred.

Although some of the aforementioned research work conceptually bears mild resemblance to the proposed research, e.g., inclination towards modularity, aspects such as soft-trialling of various re-orchestration scenarios within a virtual environment, as well as exploring the downtime associated with such re-orchestration has, to the best of our knowledge, not garnered a significant amount of attention.

While certain drawbacks such as large memory space requirement, coupled with lack of requisite programming interfaces tend to hinder implementation of SDN protocols such as Flow Sensor and Sensor Open Flow for power and memory-constrained sensor nodes [61], SDR (i.e., Software-Defined radio)-based approaches entail higher degrees of signal processing requirements rendering them equally unsuitable [93-94].

Our approach towards realization of software-defined re-orchestration too involves decoupling of functionalities from the underlying physical WSN and subjecting them to *soft-trials* or virtual re-orchestrations via software control. From the standpoint of implementation

of the re-orchestrations, the re-configurability approach has been preferred as opposed to the re-programmability as a means to attain greater flexibility, reliability and swiftness (of re-orchestrations). It ('Re-configurability-based approach) also tends to potentially allay the problem of the node going into resetting-state [37].

Re-configurability approach firstly, however, necessitates rendering the functionalities as modules that can be 'reused'. In keeping with this, it is considered reasonable to ensue upon formulation of an architectural organization wherein the key functionalities are firstly identified and subsequently *clearly defined* so as to render them as *readily-available* and reusable modules (since it can be argued that vaguely defined operational roles of the basic most constituent components seldom offer an avenue for encompassment of the softwarization paradigm). In respect to the above context, it is considered worthwhile to identify and clearly discern the three key functions (that form the building blocks of any IoT-based sensor-network organization in the underlying physical layer), namely the Leaf node function, Router node function and the Gateway node function. These well-defined core functionalities can then be included as 'modules' within a unified firmware (along with requisite auxiliary WSN modules) with which the hardware node (capable of accommodating for and executing those tasks) could be configured. The desired modules could then be flexibly enabled via external commands. These aspects are elaborated in chapter 4. In regard to the above, it is also important to clarify that owing to the advent of SoC technology, coupled with the advancements attained within the same domain, certain (advanced) SoC-based wireless microcontroller-cum-transceivers may offer adequate scope towards incorporation of softwarization paradigm.

2.4.4 NFV-WSN Merger

Despite its seemingly ineluctable prospects, limited proposals have been put forth to utilize NFV-based softwarization capability to improve the performance or to enhance programmability of wireless sensor network.

Two of the following attempts to incorporate NFV oriented approaches in WSN focus on virtualization of WSN gateways. Mouradian et al. [95] put forth an NFV-based architecture wherein the virtualization is restricted only to the WSN Gateway. The main role of this virtualized WSN gateway involve conversion of protocols and processing of information model. The network functions possess multiple VNF instances and the VNFs are stored in a centralized VNF store. Each and every VNF is managed via Element Management System (EMS) and a static chain is employed for executing them in order to deliver a service. Selection and deployment of a VNF for providing a service is undertaken by NFV Management and Orchestration (MANO) which consists of Virtualized Infrastructure Manager (VIM) as well as the VNF manager. Two separate virtualized networks are catered for by designing four VNFs. Service requests are accepted by one of the virtualized network Gateways and the required VNFs are deployed. The second virtualized network notifies the end user. Virtual Machine VM instances (i.e., particular conversions of protocols and processing of information model) of Virtualized Gateway 1 and 2 get migrated to different physical sensors employed by the authors in their setup. In another of their papers, Mouradian et al. [96] employ almost the same infrastructural framework is wherein the virtual gateway handles and operates the VNF

instances. Two separate approaches are adopted by the authors. In the first approach, the VNFs are chained within the virtual gateway and migrated to Virtual Wireless Sensor Network (VWSN) whereas in the second approach, VNFs are chained within the VWSN and are then migrated to virtual gateway. Control interfaces present within the control plane control end-to-end engagement and negotiation activities of VWSN and virtual gateway present in the application layer to ensure selection of the required parameters for deploying the correct VNF (to the end users). Luo et al. [97], on the other hand, employ NFV, in conjunction with SDN, to exert control over the sleep modes of the sensors employed in industrial WSNs as well as to manoeuvre network topology. An SDN-based logic controller unit decides upon the VNF instances to be run over the physical sensors.

2.5 Re-orchestration Latency

The latency or downtime experienced by a software defined sensor network is an interesting research topic that has been focussed upon in our research publication [26] on the proposed research topic. Herein, an example network re-orchestration scenario arising out of fragmentation event caused by departure of a ‘mobile’ router node away from the network connectivity chain is considered. The three-stage strategical approach (viz., Data Analysis and Event-Identification phase, Re-orchestration-Planning phase and the Reorchestration-Execution phase) that a network undergoes towards recovering from the network fragmentation event is laid out, indicating clearly the final of the three phases i.e., re-orchestration execution phase, as the main phase accounting for the latency experienced. That the re-orchestration latency refers to the duration of time from the instant at which the normal flow of data within the network gets disrupted (as a result of the commencement of the re-orchestration-execution process’ or phase, or due to the network fragmentation event itself) up to the instant of time of restoration of normal network-wide flow of data, is clearly specified. An estimate pertaining to such downtime could be gained beforehand in SDSN architectures encompassing virtualization environments (such as the Contiki-based Cooja) catering for soft-trials, prior to actual physical implementation. In regard to the above, it is deemed worthwhile to state that the downtime experienced by the network is influenced by host of factors viz., number of hops, number of messages exchanged amongst the nodes while the network re-orchestration process is in progress, the channel access method adopted for the network, the data communication rate set for the nodes, the topological orientation of the network, etc. Although a host of WSN deployments (typically those deployed for delay-tolerant, non-dynamic applications) may not be significantly impacted by re-orchestration process accompanied by small amount of downtime, it may cause substantial amount of loss of data in WSNs deployed for highly dynamic, latency-sensitive applications (such as those involving mobile sensor nodes).

2.6 Conclusion

A host of research works have addressed the issue of WSN flexibility in recent years [77, 78, 93, 94, 98-100]. Considerable efforts have been undertaken towards harnessing of the softwarization technologies of SDN and NFV as a means to enhance WSN flexibility both at

the node and network-levels. However, certain key operational and performance-related ‘aspects’, along with some of the pre-requisites for the same, that can be conceived to be majorly foundational towards establishing a software-defined sensor network, have not been adequately focussed and worked upon. Broadly speaking, the key operational and performance-related aspects include

- (a) development of an approach for enabling dynamic, software-defined re-orchestrations both at the node and network-levels,
- (b) establishment of a cloud-hosted library of reusable core and auxiliary firmware modules,
- (c) provisioning for trialling of such re-orchestration scenarios within a cloud-level virtual environment (whilst operating in conjunction with the library of reusable core and auxiliary firmware modules so formulated) prior to physical implementation,
- (d) outlining a strategy for the re-orchestration process, (catering for identification of events necessitating network re-orchestration, planning the necessary re-orchestrations, including extraction of real-world information for computation purposes, and finally implementation or ‘execution’ of the re-orchestration planning outcome onto the physical layer, etc), and finally,
- (e) analysis of the network downtime or latency associated with such re-orchestrations (, it being a key performance indicator of the system’s capability to react to the dynamic service requirements), etc.

The pre-requisites on the other hand (deemed logical to allow for the above) relate to

- (a) identification and definition-formalization of the core functional elements constituting any sensor network (paving the way for),
- (b) modularization of the core functional components so identified so that they can be rendered as reusable firmware modules
- (c) virtualization of the reusable firmware modules (to be harnessed for soft-trialling purposes within the virtualization environment).

The aforementioned aspects tend to form a logically structured basis towards converging upon a novel virtualization-equipped architectural framework for software-defined sensor networks that is capable of tackling dynamic sensor network operational and re-orchestration demands in a flexible manner [25].

Chapter 3

Concept Development Testbed and related Tools

3.1 Introduction

The objective of this particular chapter is to offer insight into the research methods adopted for this work as well as the hardware and software tools employed towards construction of the proposed testbed. This involves tools relevant to physical sensor network, virtual sensor network, Cloud based data management and visualization, and relevant analytical and operating system tools used for software development. The chapter justifies the relevancy of tools employed, whilst putting forth certain limitations, towards testing the experiments related to various aspects of the developed concept that is in-line with the research methods outlined in the following section.

3.2 Research Methods

The aim of the research work involves converging upon a software-defined sensor network organization that caters for its flexible re-orchestration towards coping with dynamic service requirements. This necessitates formulation of a cloud-based architectural framework consisting of both physical and virtual environments. In keeping with this, the research methods adopted herein firstly involved analyzing the literature revolving around cloud-backed WSN architectures that offer dynamic operational flexibility. Secondly, it entailed establishment of a physical WSN system by means of employing a hardware controller capable of provisioning the necessary processing memory and code storage memory to host the protocol for incrementally trialing the re-orchestration scenarios across the node-operational, node-functional, network-operational, as well as network-topological levels. Dealing with virtualization of the physical WSN so implemented so as to conduct soft-trials of the re-orchestrations (at the virtual level, so as to converge upon the most suitable ones) formed the third method of conducting this research towards solving the problems hindering dynamic WSN flexible re-orchestration.

The latter two of the aforementioned three research methods necessitated identification of the relevant software and operating system to build an Integrated Development Environment (IDE) for configuration of both physical and virtual nodes. The open source ‘Instant-Contiki-2.7’ IDE running on the Contiki OS, an operating system developed specifically for resource and power-constrained WSN nodes [101-102], presents itself as a viable tool towards implementation and testing of the proposed concept and ideas in this regard, primarily owing to two main reasons. Firstly, the embedded firmware employed for the Contiki ported-CC2538 hardware could be configured or programmed using the Contiki OS tool. Secondly, the ‘Instant-Contiki-2.7’ allows for a (WSN-specific) virtualization environment wherein virtual sensors (essentially configured with the same hardware as their physical counterparts) could be created and re-orchestrated for soft-trialing purposes (thereby ‘obviating’ physical network for testing re-orchestration scenarios).

3.3 General Organization of the Test System

The general organization of the overall system implemented for the same is as depicted in figure 3-1. The physical WSN system or testbed implemented within the lab shows the spatial distribution of the physical sensors (deployed on a ‘functional’ basis) across the premises as well as the gateway-provisioned connectivity to the remote cloud-server over the Internet. The remote cloud-server, on the other hand, hosts the necessary data storage facility and database interfaces along with configuration, virtualization, data visualization, and re-orchestration management resources.

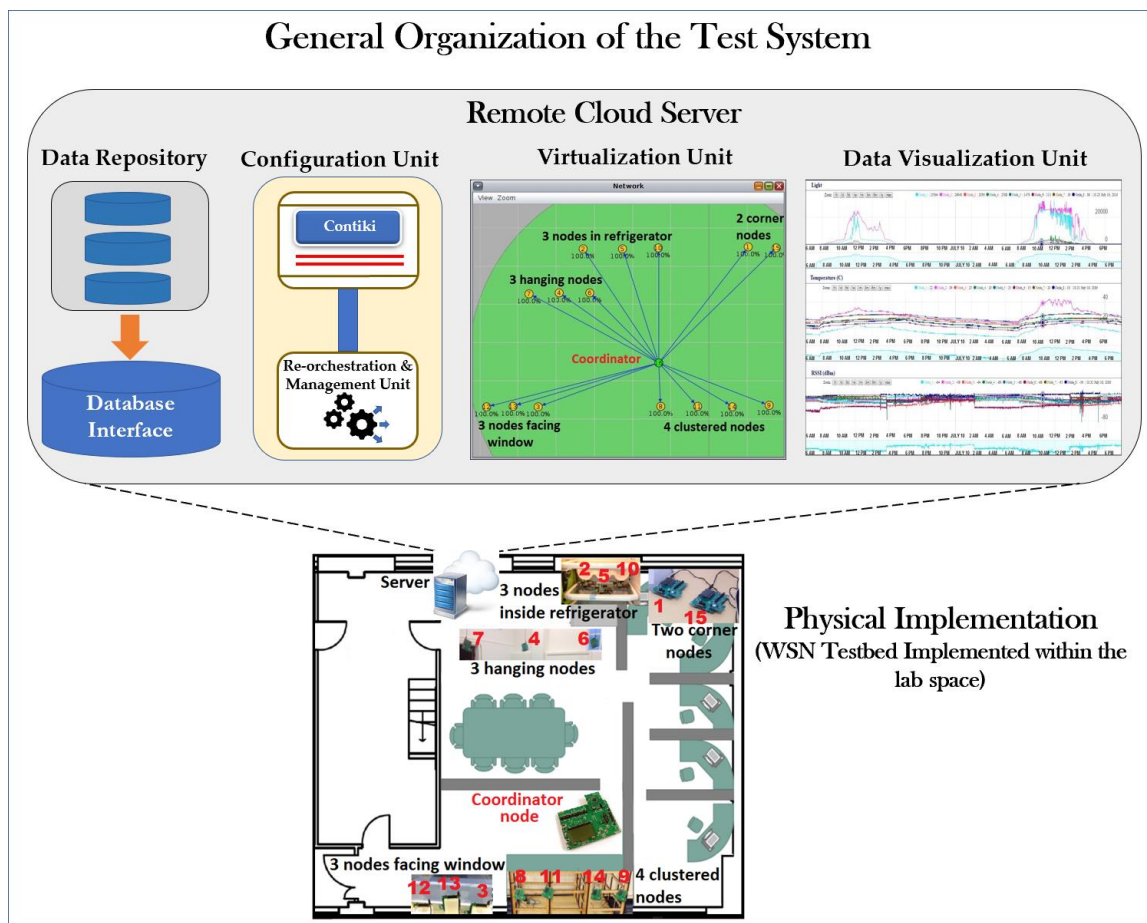


Figure 3-1 General organization of the test ‘system’.

3.3.1 Physical WSN Implementation

As shown in figure 3-2 [41-42][50], a 9-node Texas Instruments (Contiki-ported) CC2538 SoC-based (physical) sensor network was implemented within our lab premises. Herein, eight of the nine wireless CC2538 sensor-transceivers were configured using Contiki to act as end devices capturing ambient light, temperature and radio signal strength data in their respective timeslots and reporting them to a centrally deployed CC2538 serially connected to a Raspberry Pi unit which acts as the ‘Gateway’ node. By means of a python script, Raspberry Pi transmits the sensed data so received from the end devices to our local server, over the Internet.

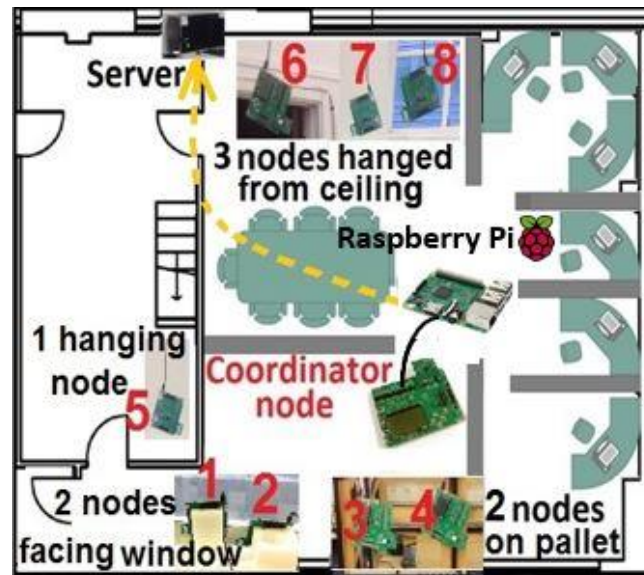


Figure 3-2 Physical implementation of a 9-node TI CC2538-based sensor network within our laboratory premises.

As depicted in figure 3-2 [41-42][50], nodes ‘1’ and ‘2’ were placed near the window to facing outside (the lab) so as to capture the ambient light during the day i.e., incident sunlight falling on it. Nodes ‘6’, ‘7’ and ‘8’ were hung towards one side of the room from the lab ceiling. Node ‘5’ too, was hung from the lab ceiling at one particular area of the lab as depicted in figure 3-2 above. Nodes ‘3’ and ‘4’ were deployed against an empty wooden pallet structure present at one particular location in the room, as shown in figure 3-2. The sensing requirements of the physical sensor network so established within the lab premises merely involved monitoring of the indoor environmental parameters of ambient light and temperature, along with the radio signal strength.

3.3.2 Remote Cloud Server Components and Associated Implementation

As alluded to earlier, the remote cloud server hosts the requisite storage, virtualization, visualization and configuration resources to ensue upon the necessary monitoring and re-orchestration demands that may be encountered by the underlying physical WSN system from time to time. It has three main components associated with it, namely, the ‘MySQL Database Server’, the ‘Webserver’ and lastly, the ‘Application Server’. The Application Server hosts the sub-components of Contiki OS, Re-orchestration Management Unit as well as the Cooja-based virtualization environment. The sub-component of Contiki IDE serves to configure both the physical and virtual sensor networks. The most suitable re-orchestrations derived from the Re-orchestration Management Unit, are directly fed to the Contiki Configuration Interface, which in turn ensues upon re-orchestration of both physical and virtual networks. Lastly, the function of the MySQL Database Server is to write queries to the database. It does so by means of PHP script which (also) fetches data obtained from the physical leaf nodes. This incoming data gets timestamped and stored in rows. A block diagram illustrating the inter-connections amongst the various constituent components of the remote cloud server is as shown in figure 3-3 [41] [50] below.

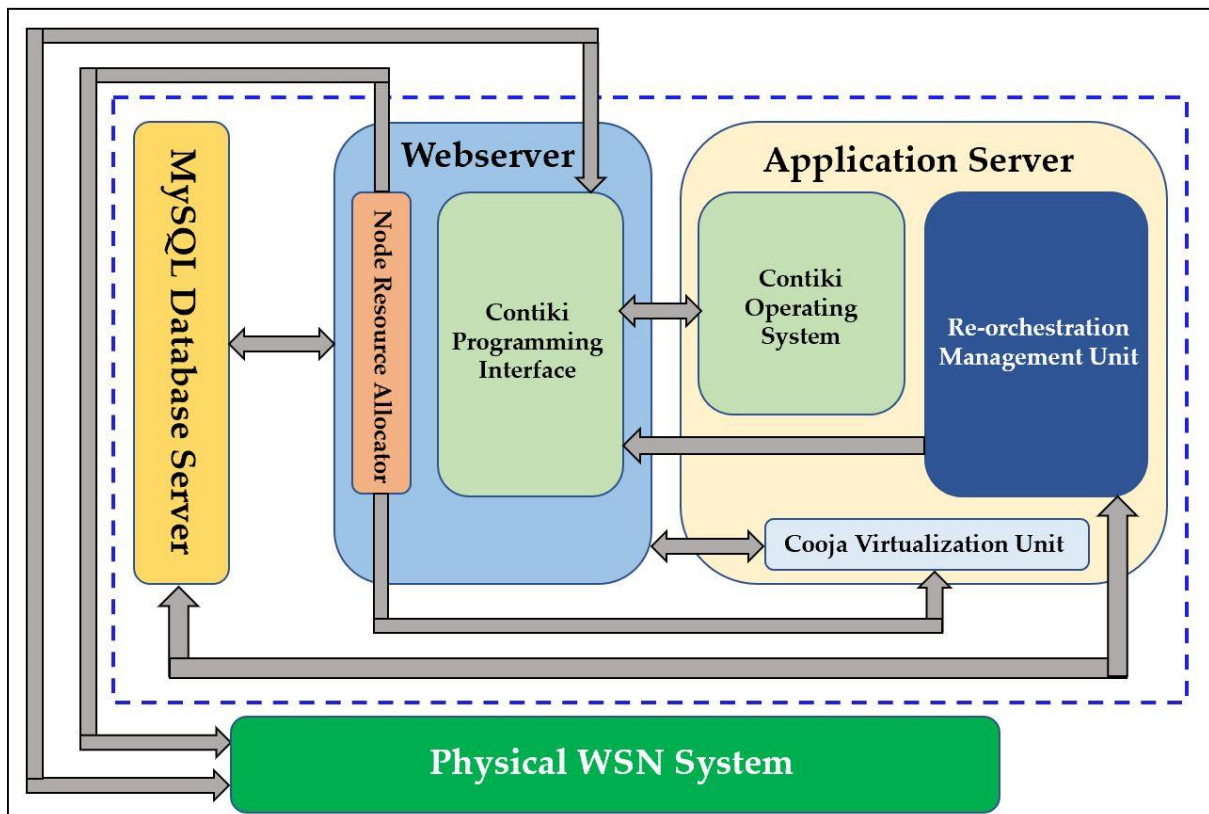


Figure 3-3 Block diagram depicting remote server implementation and the inter-relation among its various components.

Firstly, sensor data emanating from the physical leaf nodes are received by IoT Coordinator, i.e., the Raspberry Pi as depicted in figure 3-3 [41] [50], which in turn is escalated to the webservice which serves as the entry point for all the incoming sensor data. It does so by using REST API. Usage of REST API serves to exchange the necessary information between the database and the application server, besides fulfilling the important requirement of communicating with users. By means of GET and POST commands, users can select the nodes they wish to view the data for. The client side consists of the python script residing within the Raspberry Pi, which pushes the data over to the Webservice. This data gets forwarded over to the MySQL database (via REST APIs) for statistical and data processing purposes pertaining to each of the individual CC2538-based node. For scripting purposes, the PHP script is employed by the server. Optimizations foreseen within the Re-orchestration Management Unit serve as inputs to the web-server component wherein the resident Contiki configuration Interface performs the function of applying i.e., implementing them on to both virtual and physical nodes. Sensed data received within the server database get timestamped and stored in rows assigned for each individual node [50]. These data packets so received can be accessed through the PHP ‘MyAdmin’ tool.

The virtual implementation facilitated by means of the Contiki-based Cooja virtualization platform is as depicted in figure 3-4 [41-42]. Herein, Contiki has been employed to create virtual Cooja motes configured with same program code as that employed for the hardware nodes (with the exception of the functions enabling real-world sensing). One-to-one replication of all the logical facets of the operational behaviour of the virtual nodes corresponding their

physical counterparts, coupled with the similarity with respect to their arrangement within the virtualized environment, enables precise mimicking, and thereby monitoring, of the network behaviour at a virtual level (typically) hosted within the cloud.

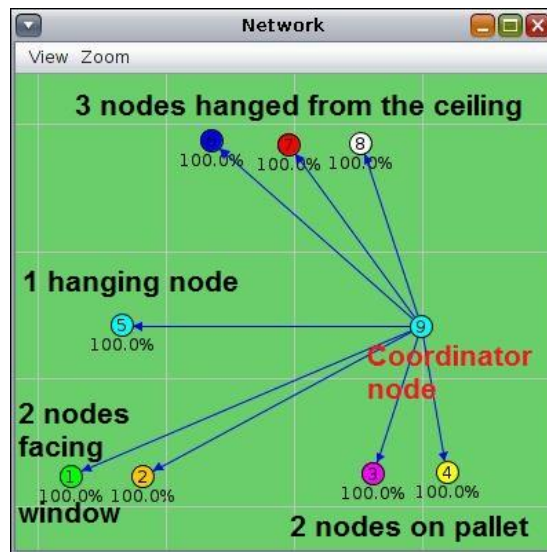


Figure 3-4 Virtual implementation of the physical 9-node TI CC2538-based sensor network within the Contiki-Cooja simulator.

Within our server, Google data visualization has been employed for Data visualization purposes. By means of a .php file residing within our server, the sensed data stored within the server database, can be viewed in graphical format (as depicted in figure 3-5 [41-42] [50]). Historical stored sensed data can be accessed for replaying a past event scenario. The graphical representation of the stored data (so collected from the physical network) provisioned via our server database is as shown in figure 3-5 [41-42] [50].



Figure 3-5 Graphical trend of sensor data (ambient light, temperature and RSSI) captured by the 8 leaf nodes, retrieved from Server database.

As is evident from figure 3-5, nodes ‘1’ and ‘2’ placed near the window facing direct sunlight during daytime reflect vastly higher ambient light sensor readings as compared to the other 6 leaf nodes. Furthermore, the same two nodes report somewhat higher temperature values as well during the same daytime period, as compared to the other leaf sensor nodes.

In regard to such trend of data as monitored via the physical sensor network, it is deemed worthwhile to assert that even under normal circumstances, considerable variations of the sensed values are observed (within a span of two days). Such an observation derived from an albeit small, but real-world WSN deployment fortifies the motivation behind driving towards a software-defined sensor network organization that possesses the requisite operational flexibility to cope and efficiently capture any significant dynamics associated with the monitored environmental phenomenon. The following sections delve deeper into the specific tools employed for this research work.

3.4 Contiki OS

Typical wireless sensor-transceiver node devices tend to be characterized by severe constraints with respect to battery capacity, memory, computational resources, etc. As a means to mitigate the impact posed by such limitations on the operational performance as well as to prolong ‘node longevity’, it is essential to efficiently manage the aforementioned scarce resources available within a typical WSN node. In this regard, selection of a suitable operating system (OS) (on which network protocols are implemented) is an important consideration. This consideration becomes even more acute in case of IoT based sensor networks wherein nodes incur the additional overhead associated with the 6LoWPAN protocol, on top of the existing network protocol, necessitating a lean OS that exerts minimum overhead on the low power sensor-cum-transceivers. As mentioned earlier, Contiki is an operating system specifically developed for the wireless microprocessors [102]. The Instant Contiki IDE offers support for firmware development for a large number of different low power wireless hardware target platforms viz., 8051, ARM Cortex- M3, MSP 430, AVR, z80, etc., [102], thereby enabling the possibility of formulating heterogenous networks, if required.

From this perspective of this research work, an operating system, endowed with desirable characteristic features such as real-time scheduling capability, portability, whilst being lightweight, flexible [102] and open source, is well suited for the hardware nodes to run on.

The Contiki Operating system fulfils the aforementioned characteristic requirements owing to

- enabling dynamic program loading (as well as unloading) and concurrency of node tasks (whilst operating in complex, dynamic and concurrent IoT-based sensor network environments).
- exerting lesser memory overhead, with RAM and ROM requirements of 2 kB and 40 kB respectively [103], on the target hardware platform for which it is employed.
- Employing the lightweight, hybrid programming model of ‘protothreads’ (combining the multi-threaded programming-based model, along with the ‘event-driven’ mechanism) which allows for ‘conditional-blocking’ wait-based operation, paving the way for sequential control flow [104-105].

Developed as an open source, readily available OS specifically for IoT-based low power WSN devices, Contiki OS offers support for the requisite IP standards of 6LoWPAN, CoAP, RPL, etc. It also encompasses a thin-layered stack of lightweight communication protocols known as the RIME stack allowing for code reuse and simplification of network protocol implementation [106]. Numerous large-scale, real-life implementations have employed Contiki OS [107] for applications such as wildlife monitoring, intrusion detection, surveillance, road tunnel fire monitoring, etc.

3.4.1 Comparison of Contiki with other Operating Systems

Comparison amongst the various operating systems used for WSN nodes (with respect to parameters such as memory requirements, real time support, protocols, etc.) has been provided in Table 3-1.

Table 3-1 Comparison table of operating systems used in WSNs.

Parameters	Operating Systems							
	Tiny OS	Contiki	Linux	RIOT	Mantis	Nano-RK	t- Kernel	SOS
Minimum ROM Memory	3.4 kb	3.8 kb	~ 1MB	~ 5kb	14 kb	10 kb	28.2 kb	20 kb
Minimum RAM Memory	230 bytes	< 2 kb	~ 1MB	~ 1.5 kb	500 bytes	2000 bytes	2000 bytes	1.16 kb
C Support	No	Yes	Yes	Yes	Yes	Yes	No	Yes
Modularity	Yes	Partial support	No	Yes	Yes	Yes	_____	Yes
Real Time Support	Yes	Yes	No	Yes	No	Yes	Yes	No
MAC Protocol	B-MAC, S-MAC	Contiki MAC	TDMA MAC	-	X-MAC	B-MAC & CSM A-CA	Yes	-
Other Protocol	RPL, CTP	μ IPv6, RPL	-	-	-	Zigbee and 'LEE AND'	Yes	SOS
Wireless Re-programming	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Dynamic Memory	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes

Contiki OS tends to serve as a better alternative (OS) when low power nodes such as the Texas Instruments ‘CC2538’ SoCs need to undergo frequent firmware updates for IP-based experiments. Moreover, Contiki supports dynamic replacement of a pre-existing application within a given node’s firmware (as a means to update it). This feature is not supported by Tiny OS, wherein the application along with the operating system has to be replaced completely [108].

Contiki OS is a lightweight and flexible OS which is specifically designed for the power and memory-constrained WSN nodes. It consists of its own layered NETSTACK, which has been devised to fulfil the requirements of IoT nodes. It is emerging as one of the preferred OS for IoT devices owing to some of the desirable characteristics encompassed by it viz., portability, concurrency, low memory occupancy, real-time support, C-language support (for programming), etc., besides being readily available (i.e., an open-source OS). Besides offering support for 6LoWPAN, it consists of the custom MAC protocol of ContikiMAC, and the low-power IETF-RPL which tends to ensure node-lifetime longevity [109] making it highly suitable to be employed as an OS for both physical and virtual WSN implementations in the proposed project.

3.4.2 Contiki Netstack

The Contiki Netstack is a layered stack of protocols specifically designed towards overcoming the shortcomings associated with the conventional OSI model in fulfilling the requirements pertaining to the IoT domain [110]. The various protocol implementations (files) available across the various layers of the Contiki are as shown in figure 3-6 [111].

The protocol modules of ‘cc2538-rf.c’ for ‘Radio’ layer, ‘nullrdc.c and contikimac.c’ for ‘Duty Cycling’ layer as well as ‘csma.c’ for ‘MAC’ layer are available at the lower three layers of the stack are utilized for custom-configuring the ‘Radio’, ‘Duty Cycling’ and ‘MAC’ layers [111]. Roussel and Song [112] state that separation of layers forms the basis of the stack design. This is evident from the partitioning of the MAC layer into two separate layers of MAC and Radio Duty Cycling (RDC), even though the state-of-the-art MAC protocols are meant to manage aspects pertaining to both the layers.

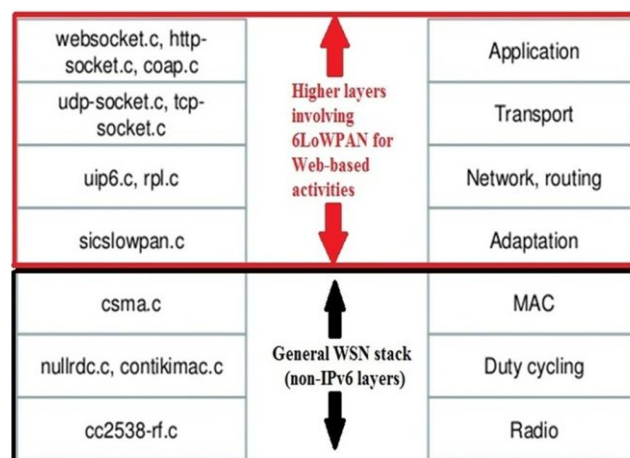


Figure 3-6 Contiki Netstack depicting protocols available across the various layers.

3.5 Texas Instruments CC2538 Target Hardware

The physical layer of a Cloud-governed, IoT-based sensor network is generally composed of a number of wireless sensor-transceiver target platforms, typically configured with either the sensing, routing or gateway functionalities. Communication operations executed by these hardware nodes are defined by the protocol with which they are configured. Keeping in mind the dynamic service requirements a physical sensor network needs to comply with, a plethora of factors need to be taken into account whilst converging upon a suitable device for setting up an IoT-based physical sensor network testbed. These include current consumption, sensing capability, OS, processor or Micro-Controller Unit (MCU) features, memory size, protocols supported, cost, IoT capability, logical compliancy with corresponding virtual entity, form factor, etc.

The Contiki-ported (natively supported) Texas Instruments CC2538 Evaluation Module (EM) (SoC mote), as shown in figure 3-7, is an advanced wireless transceiver with IP-configurability. It consists of the 32-bit ARM Cortex M3 microcontroller providing for memory requirements of up to a maximum of 32 KB and 512 KB of RAM and flash respectively [113]. Its salient features include small footprint, low dynamic current consumption, low cost [114] and various power modes that could be leveraged upon to prolong ‘node’ lifetime (via conserving battery power by dropping down to a low energy mode or sleep mode when appropriate). It supports ZigBee IP Mesh, ZigBee PRO Mesh, ZigBee RF4CE, advanced ‘ZigBee profiles’, 6LoWPAN, and 2.4 GHz 802.15.4-based solutions. It consists of the on-chip ‘temperature’ and RSSI (Received Signal Strength Indication) sensors. Apart from the above, it exhibits excellent receiver sensitivity and programmable output power. Such highly desirable features (e.g., the support for Zigbee protocol that was employed for setting up of the physical testbed within the lab premises) render it to be highly favourable for physical sensor cloud (PSC) testbed implementation.



Figure 3-7 CC2538 Evaluation Module.

The CC2538 EM is used in conjunction with the SmartRF 06 Board (as shown in figure 3-8) for programming purposes. It consists of the two on-board sensors of ‘Accelerometer’ and the ‘Ambient Light Sensor’.



Figure 3-8 SmartRF 06 Evaluation Board.

3.5.1 Power Modes and Current Consumption Profile

CC2538 motes are characterized by low current consumption and can be switched to different operational modes (to conserve power depending upon the prevailing operational requirements). Current consumption details of two of the sleep modes, as specified (by Venkatesh, G.) in [115], have been summarized in ‘Table 3-2’ below.

Table 3-2 ‘Current consumption’ values of TI CC2538 for the two sleep modes.

Power modes	Current consumption [115]
PM2	1.3 μ A
PM3	0.4 μ A

In an attempt to study the current consumption of the CC2538 motes, the current profile consumption profile for the CC2538 hardware, operating in different power modes was obtained via oscilloscope as shown in figure 3-9 below. Herein, initially the CC2538 mote is subjected to the power mode PM2 after which it was woken up by means of an external interrupt i.e., hard reset (via pushing the ‘RESET’ button on the CC2538 Evaluation Module). Upon ‘waking up’, it momentarily enters an ‘idle state’ before executing the transmission and reception operations in the ‘active state’. Once these (requisite) operations within the ‘wake-up’ period are executed, the node is again subjected to deep sleep PM2 mode to conserve battery power.

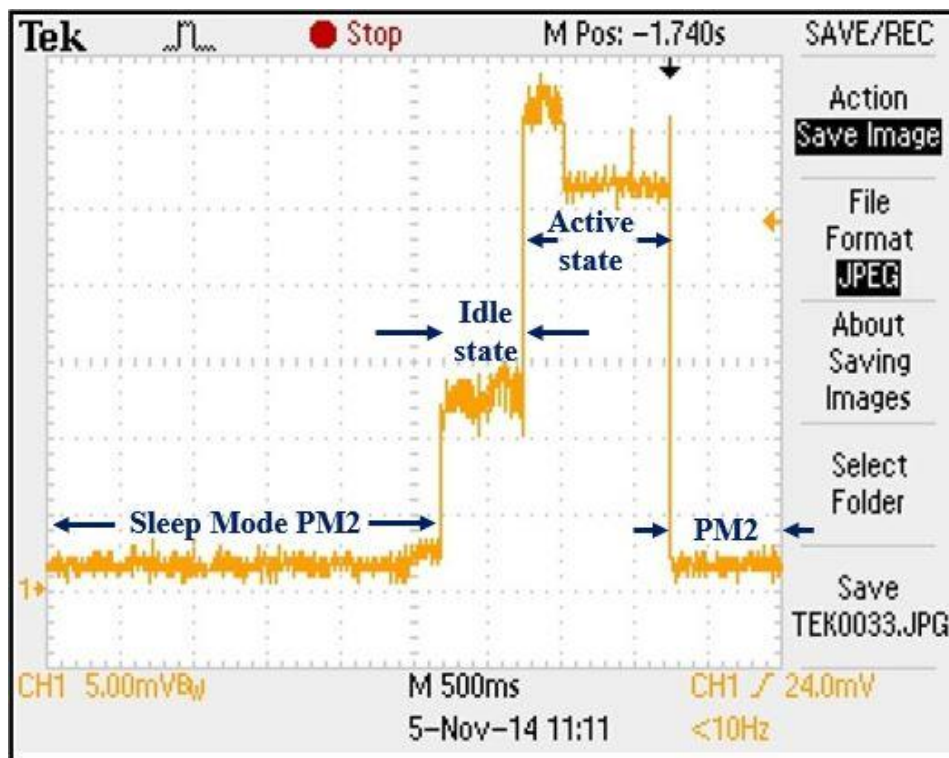


Figure 3-9 Current consumption profile of TI CC2538 hardware obtained via oscilloscope.

3.5.2 Contiki as a Tool for TI CC2538 Configuration

Contiki IDE forms the environment for the development of the firmware with which the CC2538 chips have been configured. By means of writing programs in ‘C’ language, the nodes could be configured with the Leaf sensing, Routing or the IoT-based gateway functionalities. Owing to certain key enhancements allowing for ample memory, processing computation capabilities, etc., coupled with a (robust and) powerful microcontroller unit (MCU), the CC2538 devices could either be pre-configured (via Contiki-based software control) to execute single or multiple functionalities at a time or dynamically re-orchestrated via software control to assume or repeal additional functionalities. Reason being, Contiki allows for development the ‘modular’ firmware i.e., a firmware designed in such a way that it consists of numerous requisite ‘software modules’. Each such constituent software module executes a certain given function or task (owing to encompassment of the necessary variables and source code), allowing for reusability of code. This in turn opens the door for adoption of the reconfigurability approach. In addition to this, Contiki IDE provisions for necessary reconfiguration of the various node-operational parameters available for manipulation at the various layers within the communication stack viz., sensor selection, channel allocation, (sensed) data buffering, radio transmission power, channel access method, etc. (elaborated in chapter 5). Using Contiki-based software configuration control as a service, the CC2538 motes used in this research can be configured as IP-configured or non-IP configured motes, by means of enabling or disabling the 6LoWPAN protocol module, respectively, available within it.

Contiki IDE facilitates for the compilation and error checking for the C program written within it. Prior to configuring the CC2538 hardware platform with the successfully compiled ‘C’ code, the ‘Uniflash’ software installed within it converts into the requisite executable

format i.e., the ‘.elf’ (Executable Linkable Format) format. As a standalone SoC, the TI CC2538 EM consists of the on-chip ADC (Analog-to-Digital) temperature and RSSI. When used in conjunction with the SmartRF06 board (i.e., when the EM is mounted over the SmartRF06 board with the help of the EM pin connections), the CC2538EM can access the Osram SFH 5711 ‘Ambient Light Sensor’ present on the SmartRF06 board. By means of inclusion of the necessary header files as well as respective sensor functions within the Contiki-based ‘C’ programs so written and compiled to configure the CC2538 SoCs as leaf nodes, three sensor variables have been extensively used for physical data collection and testing purposes within this research.

In real-life sensor network applications, especially the ones involving mobility, RSSI sensor-based data could be of high importance. Events wherein a mobile router node happens to venture beyond the communication range of its children or neighbouring router nodes could result in isolation of that respective part of the network. In this regard, a real-life test, albeit incremental, was performed within our lab premises using CC2538 chips with the intention of testing the RSSI values reflected by a mobile router node with respect to a stationary leaf node. Table 3-3 depicting values recorded for the departing router as its distance from the stationary leaf node increases is as shown below.

Table 3-3 Real RSSI values recorded for the departing router (with respect to the stationary leaf node) via physical experimentation.

Distance of the departing router with respect to the Gateway (m)	RSSI value of the departing router with respect to the Gateway (dBm)
1	-48
2	-57
3	-59
4	-61
5	-65
6	-66
7	-78 (Outside lab premises)

Although the light, RSSI and temperature sensor values captured by the TI CC2538 sensors may not be precisely accurate, they can be used for nominal demonstration and research prototyping purposes.

3.6 Raspberry Pi SoC

The inexpensive and re-configurable single-board computer of Raspberry Pi SoC, as depicted in figure 3-10, endowed with Internet connectivity capability, is a widely used component that can be seamlessly integrated within the sensor network applications’ domain for research and prototyping purposes [116]. A Raspberry Pi board consist of RAM memory, the CPU (Central Processing Unit), power supply connector, USB ports (wherein the Wi-Fi USB dongle or adaptor can be attached), ethernet ports, GPU (Graphics Processing Unit), the lower-level peripherals of General Purpose Input Output (GPIO) pins that can be utilized for

I2C, UART, SPI-based serial communication buses or interfaces, etc. Advancements over the recent years have resulted in a number of enhancements with respect to memory capacity, CPU, GPU, etc. (i.e., in the Raspberry Pi 4 version) [116]. It consists of a slot for an SD card, enabling it to be used either for storing a large amount of data or for datalogging purposes (in cases of Internet outage). Experience available within the lab rendered Raspberry Pi as an automatic choice towards its employment as a gateway. This, coupled with provisions for being battery powered, renders it to be used as a portable device. A host of operating systems can be used to run Raspberry Pi SoCs such as Raspbian, NOOBS, etc.



Figure 3-10 Raspberry Pi SoC.

Being equipped with both Wi-Fi and ethernet-based connectivity to access the Internet, coupled with ample computational and processing power, the Raspberry Pi SoC has been employed as a gateway within our research wherein it performs its function as a protocol-converter and transmits the CC2538-based sensed data over the Internet to the remote server.

3.7 Contiki-based Cooja as a Virtualization Platform

Advancements in software and modelling technologies have ushered the development of tools such as Contiki that enable same programs to be used for both hardware and simulation environments. Contiki's in-built java-based tool of 'Cooja' [117-118] (present within 'Instant-Contiki 2.7 IDE) can be used for virtualization of IoT based networks both at the hardware as well as the 'less detailed' levels. Within our research, Cooja serves as a virtualization platform wherein the 'Contiki OS' generated firmware employed for compiling and configuring the physical target hardware platform of TI CC2538 are the ones used for compiling and creation of virtual nodes. This one-to-one reciprocity with respect to the configuration of the nodes enables an accurate virtual representation of the logical (software code) facet of the operational dynamics of the physical environment.

It is deemed worthwhile to allude to the saliciencies as well as limitations when employing Cooja as a virtualization tool for this research work. Since this research work entails usage of the same Contiki-generated software codes for ‘compiling’ and configuring both the virtual Cooja motes as well as the physical TI CC2538 nodes (as alluded to in chapter 3), re-orchestrations with respect to

- (certain) logical facets of operational behaviour of the physical node (by means of wresting software-defined control over the Contiki firmware fed into it) such as the aforementioned flexibilities (viz., sampling rate, buffer size, MAC layer manipulations, etc.),
- node-level functional re-orchestration from an existing function to a different function (e.g., leaf to router) via switching over to that software

to be imposed on the underlying physical layer consisting of (Contiki-ported CC2538 hardware nodes) can be accurately represented, trialled and analysed within the virtual environment beforehand via Contiki-based software control. By means of providing an avenue for such precise and unambiguous replication of logical facets of the operational behaviour and re-orchestrations, Cooja serves as an as an extremely desirable tool for virtualization.

3.7.1 Node Mobility

Cooja supports node mobility. Upon patching the mobility plugin within Cooja, it is possible to exert fine tune control over the traversal paths as well as speed of a particular virtual node or a group of nodes. Using a re-writable or editable ‘positions.dat’ file, the coordinates as well as the corresponding time-instants of a node or a group of nodes can be pre-defined with Cooja to set the path (in terms of coordinates and hopefully speed, if possible) of a particular mobile node.

3.7.2 In-built RSSI Model

Cooja consists of an in-built RSSI or ‘Radio Propagation’ model by means of which the radio signal strength of a particular node with respect to another node can be obtained. This feature (denoted as Radio) within Cooja is available within the ‘Mote Interface Viewer’ option that can be accessed via right clicking on any of the motes created. The RSSI values obtained from this model are mostly distance dependent and the practicalities associated with real-world are not accounted for within this model.

Consider the following example wherein a virtualized (2-node) point-to-point network has been implemented within Cooja. The purpose of this example is two-fold, reflecting the ‘built-in radio propagation model’ as well as the ‘mobility’ features offered by the Cooja virtualization tool. Herein, node 1 is a mobile node configured with the router function whereas node 2 has been configured with the leaf function which transmits the data packets sensed by it to the departing router (till the time the mobile router is within its communication range). Once the mobile router moves beyond the communication range of the leaf node, it goes without saying that no communication ensues between the two, even though the leaf node may continue to transmit data packets meant for the (departed) router.

The mobile router node traverses through a pre-meditated path of traversal (as per by the coordinates and their corresponding time instants entered within the ‘positions.dat’ file). Figure

3-11 a depicts the (initial) case when the two nodes are in (close) proximity of each other. The RSSI value of the mobile router leaf node 1 with respect to node 2 (as obtained from the radio propagation model feature) for this case is as depicted in figure 3-11 b.

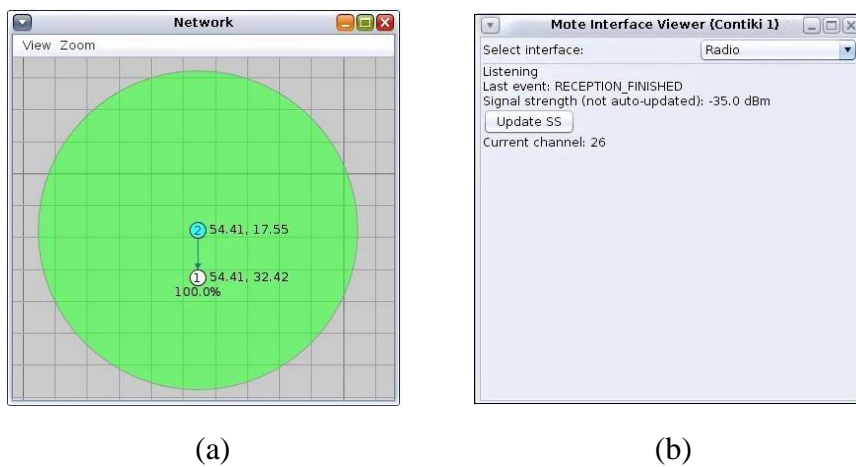


Figure 3-11 Radio propagation model feature within Cooja (a) Network window within Cooja wherein the mobile router and the stationary leaf nodes are initially in close proximity of each other; (b) ‘Mote Interface Viewer’ window within Cooja wherein the RSSI values of the mobile router with respect to the leaf node can be viewed (within the ‘Radio’ option in its dropdown list).

As the distance between the two nodes increases owing to departure of the leaf node, the radio signal strength between the two nodes degrades as per the built-in propagation model, as depicted by the figures 3-12 a and b.

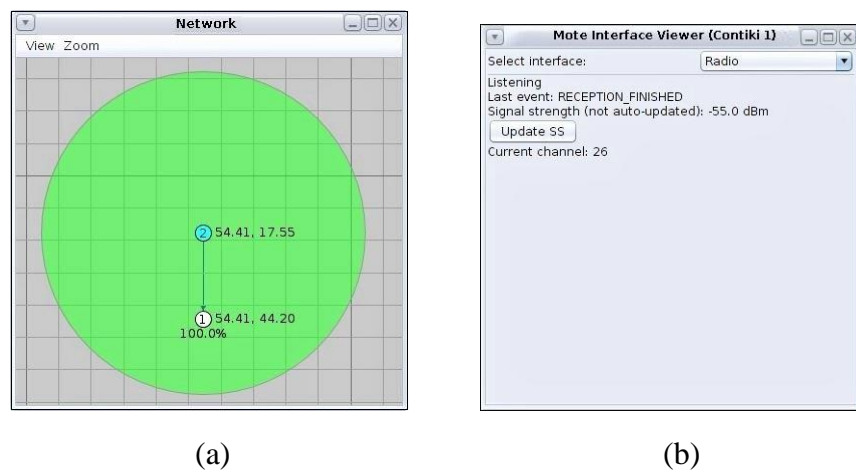


Figure 3-12 Radio propagation model feature within Cooja (a) Network window within Cooja wherein the mobile router has moved away from the stationary leaf node to a certain extent; (b) ‘Mote Interface Viewer’ window within Cooja wherein the degraded RSSI value of the mobile router with respect to the leaf node (as compared to the earlier case) can be viewed (within the Radio option in its dropdown list).

Upon further increasing the distance between the two nodes as shown in figure 3-13 a, the RSSI signal value of node 1 with respect to node 2 drops further, as reflected by the in-built model (shown in figure 3-13 b).

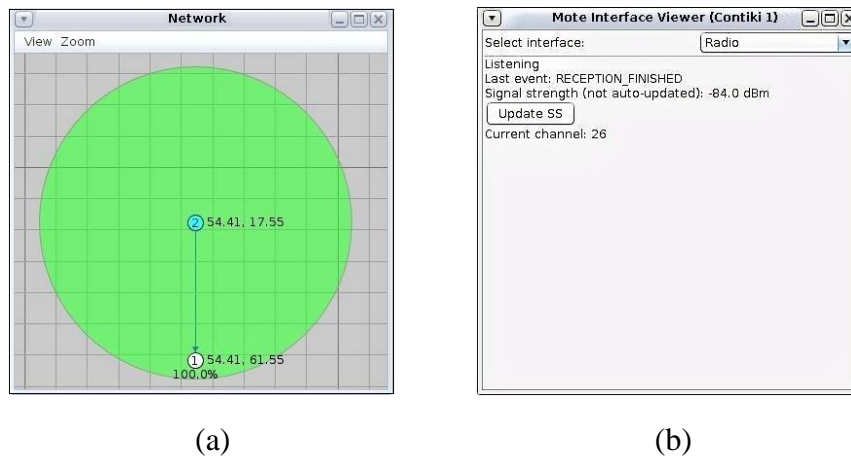


Figure 3-13 Radio propagation model feature within Cooja (a) Network window within Cooja wherein the mobile router has moved quite far away from the stationary leaf node but is still within its communication range; (b) ‘Mote Interface Viewer’ window within Cooja wherein the further degraded RSSI value of the mobile router with respect to the leaf node (as compared to the earlier case) can be viewed (within the ‘Radio’ option in its dropdown list).

Once the departing leaf node ventures out of the communication range of the router (as can be seen from figure 3-14a), radio communication messages between the two nodes cease to transpire. The in-built radio propagation model reflects a value of -100 dBm as depicted in figure 3-14 b.

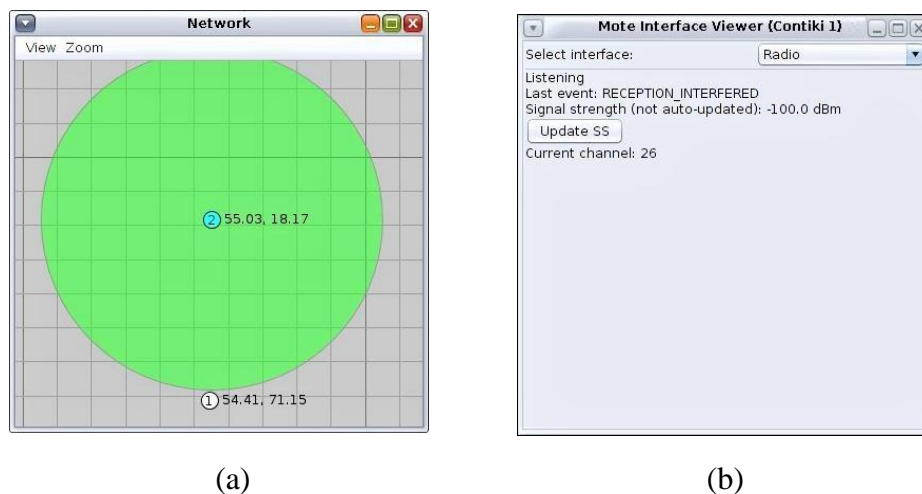


Figure 3-14 Radio propagation model feature within Cooja (a) Network window within Cooja wherein the mobile router has moved beyond the communication range of the stationary leaf node resulting in loss of connectivity; (b) ‘Mote Interface Viewer’ window within Cooja wherein RSSI value (of the mobile router with respect to the leaf node) pertaining this particular case of loss of connectivity is reflected by an RSSI value of -100dBm (as viewed within the ‘Radio’ option in its dropdown list).

The provision of begetting instantaneous values of RSSI signal of one node with respect to another is a handy feature available within the open-source tool of Cooja, which could prove to be somewhat beneficial both in terms of setting up the virtual implementation (of a real-life physical network) in for remote monitoring purposes, as well as during running soft-trials of network-re-orchestration scenarios within it.

3.7.3 ‘Power-trace’ Feature

Another feature associated with Cooja is the software-based ‘power-profiling’ mechanism of ‘Powertrace’ which keeps track of the estimated energy ‘expenditure’ of each sensor node without any additional hardware requirement. The ‘Powertracker’ feature within Cooja can be accessed from within the Tools’ menu dropdown list. It provides instantaneous ‘Radio TX’ (i.e., Radio Transmission), ‘Radio RX (i.e., Radio Reception)’ and ‘Radio on’ values of the constituent virtual nodes as well as the ‘average’ of the overall network, as percentages.

The ‘Powertrace’ feature has been elucidated by means of an example experiment conducted within Cooja. Herein, a 5-node virtual star-topological network consisting of one router-coordinator and 4 leaf nodes are initially configured to operate under a CSMA (Carrier Sense Multiple Access) scheme, as depicted in figure 3-15a. The sampling rate of the four of the leaf nodes was deliberately set to a very high value of 250 samples per second. From figure 3-15b, it is clear that the ‘RX’ and ‘TX’ values (in terms of percentages) of the constituent nodes are considerably high.

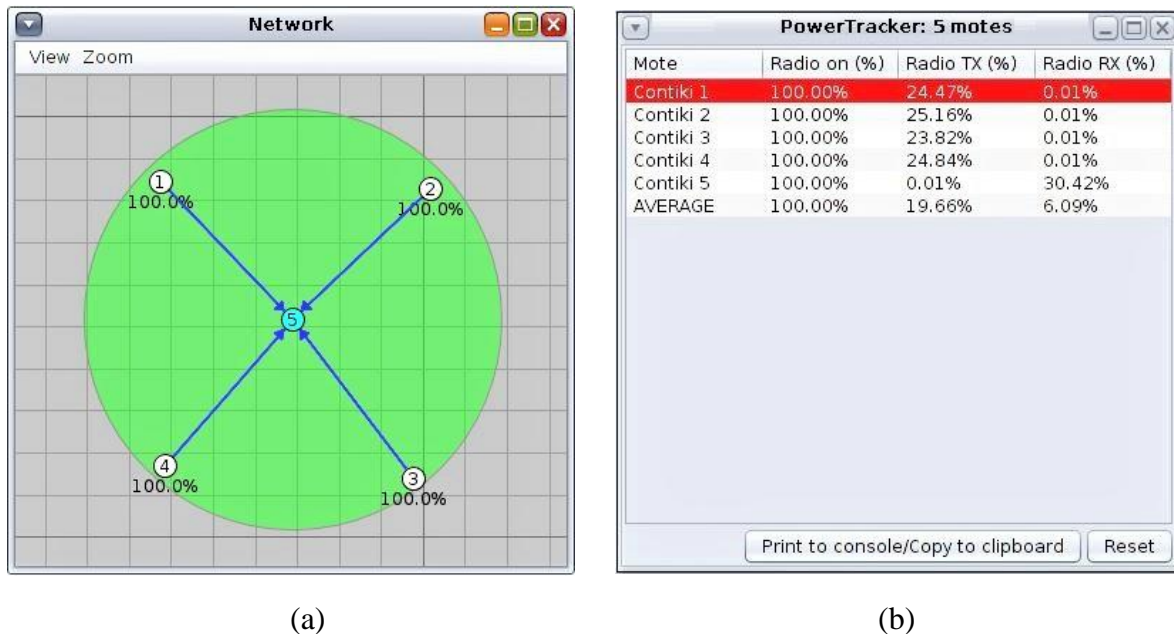


Figure 3-15 Powertrace feature within Cooja (a) Network window within Cooja showing a ‘star topology’ -based 5-node virtual network within Cooja operating under CSMA mode and (b) Screenshot of the ‘Powertracker’ window showing extremely considerably high ‘Radio On(%)’, ‘Radio RX(%)’ and ‘Radio TX(%)’ values.

Upon re-orchestrating the network to operate under the TDMA scheme, it was observed that the percentage values of ‘RX’ and ‘TX’ of the constituent nodes were extremely minimal, as can be seen from figures 3-16 a and 3-16b.

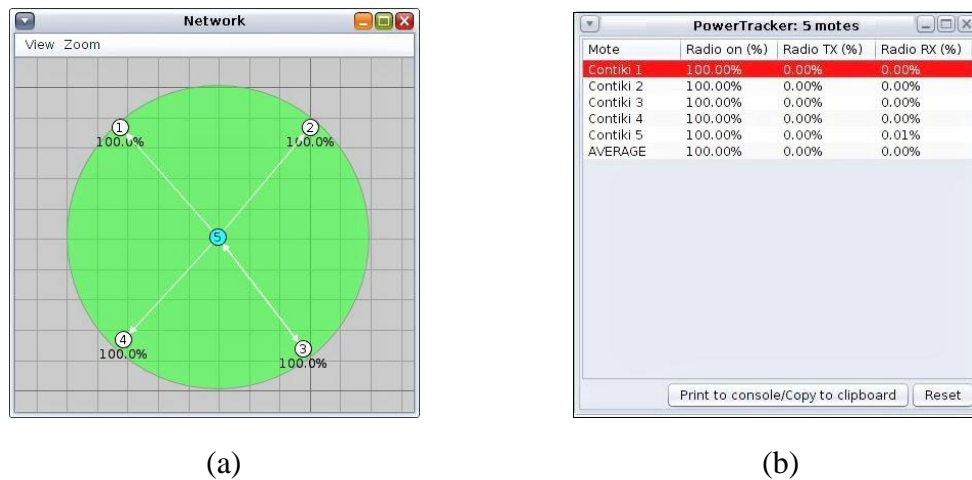


Figure 3-16 Powertrace feature within Cooja (a) Network window within Cooja showing the same ‘star topology’ -based 5-node virtual network within Cooja operating under TDMA mode and (b) Screenshot of the ‘Powertracker’ window showing extremely minimal ‘Radio On(%)’, ‘Radio RX(%)’ and ‘Radio TX(%)’ values.

By means of serving to indicate the power consumed by the nodes, the ‘Powertrace’ feature obtained from Cooja could be useful for gaining an estimate of the (power-consumption-based) performance characteristics of the (nodes and the entire) network (, as a whole).

3.7.4 Provisioning for Virtual 6LoWPAN-Based Network

UIPv6 stack within Cooja provides IPv6 networking. Cooja offers support (in terms of creation of virtual nodes) for multiple target hardware platforms, one of which is the Sky mote. The following examples illustrates how Cooja can be utilized to create a virtualized 6LoWPAN-based network consisting of one ‘Border-router’ (or ‘Edge router’ node) and four ‘Er-example-server’ nodes, as shown in figure 3-17.

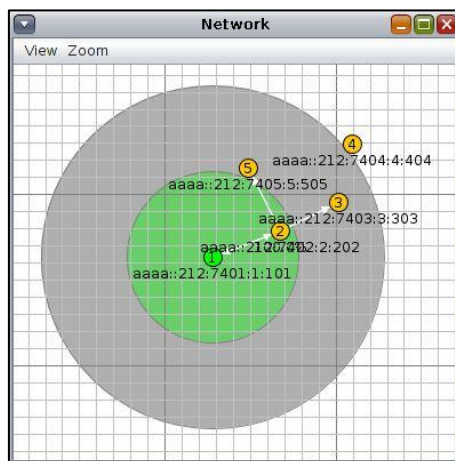
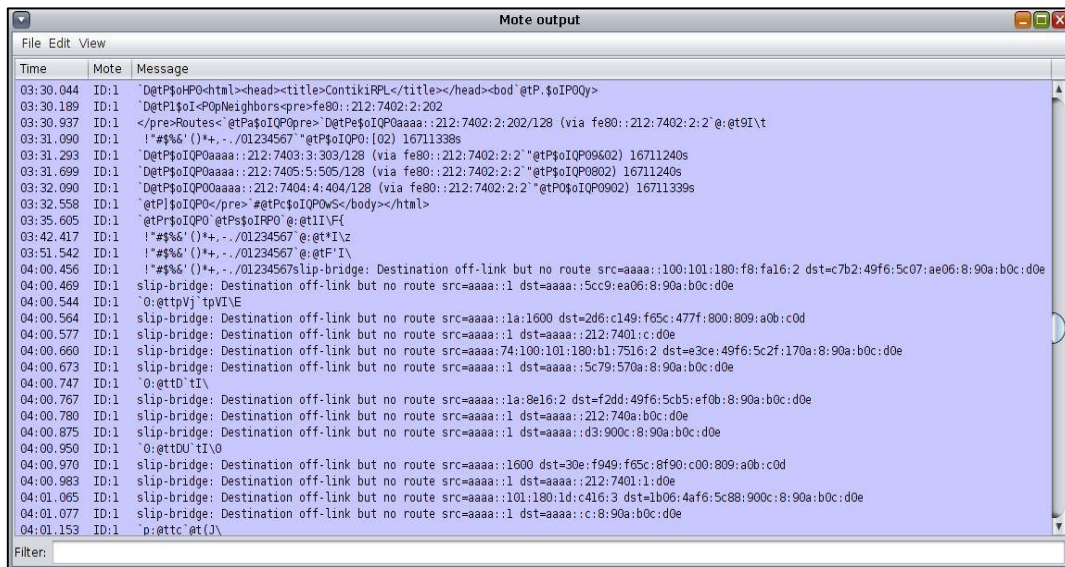


Figure 3-17 6LoWPAN-based virtual network consisting of one border router node and 4 ‘Er-example server’ nodes created within Cooja.

Herein, the Edge router or border router node (created using SKY mote also supports 6LoWPAN), acts as a bridge between 802.15.4 network and the Internet (facilitating for serial

to Internet interface). The four virtual ‘Er-example-server’ nodes (represented by the yellow colour motes) have been configured using the ‘Er-example-server.c’ program code. Figure 3-18 depicts the ‘mote output’ within Cooja showing the addresses of the motes are getting printed.

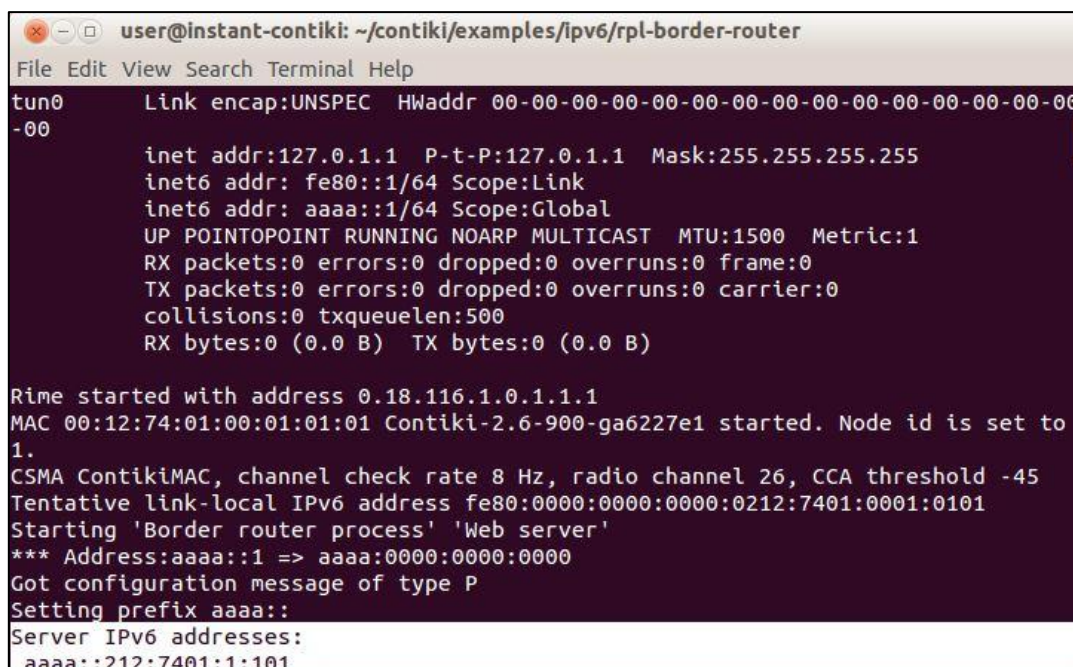


```

Time      Mote      Message
03:30.044 ID:1    `DetPfoHP0-dhtml>-head<title>ContikiRPL</title></head><body>@tP,foIP00y>
03:30.189 ID:1    `DetP1foI<PpNeighbors-spre>fe80::212:7402:2:202
03:30.937 ID:1    </pre>Routes<`@tPfoIP0pre>`DetPfoIP0aaaa::212:7402:2:202/128 (via fe80::212:7402:2:2`@t9I\t
03:31.090 ID:1    !`#%`'()+,./01234567`@tPfoIP0:02) 16711339s
03:31.293 ID:1    `DetPfoIP0aaaa::212:7403:3:303/128 (via fe80::212:7402:2:2`@tPfoIP09602) 16711240s
03:31.699 ID:1    `DetPfoIP0aaaa::212:7405:5:505/128 (via fe80::212:7402:2:2`@tPfoIP0802) 16711240s
03:32.090 ID:1    `DetPfoIP0aaaa::212:7404:4:404/128 (via fe80::212:7402:2:2`@tPfoIP0902) 16711339s
03:32.558 ID:1    `@tPfoIP0</pre>`#@tPcfoIP0wS</body></html>
03:35.605 ID:1    `@tPfoIP0`@tPfoIPPO`@:etI\F{
03:42.417 ID:1    !`#%`'()+,./01234567`@:etI\z
03:51.542 ID:1    !`#%`'()+,./01234567`@:etF\I
04:00.456 ID:1    !`#%`'()+,./01234567slip-bridge: Destination off-link but no route src=aaaa::100:101:180:f8:fa16:2 dst=c7b2:49f6:5c07:ae06:8:90a:b0c:d0e
04:00.469 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::5cc9:ea06:8:90a:b0c:d0e
04:00.544 ID:1    `o:ettpVj`tpVI\E
04:00.564 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1a:1600 dst=2d6:c149:f65c:477f:800:809:a0b:c0d
04:00.577 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::212:7401:c:d0e
04:00.660 ID:1    slip-bridge: Destination off-link but no route src=aaaa:74:100:101:180:b1:7516:2 dst=e3ce:49f6:5c2f:170a:8:90a:b0c:d0e
04:00.673 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::5c79:570a:8:90a:b0c:d0e
04:00.747 ID:1    `o:etD`tI\O
04:00.767 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1a:0e16:2 dst=f2dd:49f6:5cb5:ef0b:8:90a:b0c:d0e
04:00.780 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::212:740a:b0c:d0e
04:00.875 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::d3:900c:8:90a:b0c:d0e
04:00.950 ID:1    `o:etDU`tI\O
04:00.970 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1600 dst=30e:f949:f65c:8f90:c00:809:a0b:c0d
04:00.983 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::212:7401:1:d0e
04:01.065 ID:1    slip-bridge: Destination off-link but no route src=aaaa::101:180:1d:c416:3 dst=1b06:4af6:5c88:900c:8:90a:b0c:d0e
04:01.077 ID:1    slip-bridge: Destination off-link but no route src=aaaa::1 dst=aaaa::c:8:90a:b0c:d0e
04:01.153 ID:1    `o:ettc`et(J)\
  
```

Figure 3-18 Mote output window of the 5-node 6LoWPAN-based virtual network within Cooja showing the addresses of the constituent motes are getting printed.

The terminal window within Contiki wherein the server IPv6 address is getting printed is as shown in figure 3-19.



```

user@instant-contiki: ~/contiki/examples/ipv6/rpl-border-router
File Edit View Search Terminal Help
tun0      Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
-00

inet addr:127.0.1.1 P-t-P:127.0.1.1 Mask:255.255.255.255
inet6 addr: fe80::1/64 Scope:Link
inet6 addr: aaaa::1/64 Scope:Global
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Rime started with address 0.18.116.1.0.1.1.1
MAC 00:12:74:01:00:01:01:01 Contiki-2.6-900-ga6227e1 started. Node id is set to
1.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26, CCA threshold -45
Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7401:0001:0101
Starting 'Border router process' 'Web server'
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
aaaa::212:7401:1:101
  
```

Figure 3-19 Terminal window within Contiki wherein the server IPv6 address is getting printed.

Successful establishment of connection is verified by means of issuing the ‘ping6’ command, as shown in terminal window depicted in figure 3-20.

```

user@instant-contiki: ~/contiki/examples/ipv6/rpl-border-router
File Edit View Search Terminal Help
user@instant-contiki:~/contiki/examples/ipv6/rpl-border-router$ ping6 aaaa::212:7401:1:101
PING aaaa::212:7401:1:101(aaaa::212:7401:1:101) 56 data bytes
64 bytes from aaaa::212:7401:1:101: icmp_seq=1 ttl=64 time=7047 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=8 ttl=64 time=128 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=9 ttl=64 time=15.1 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=10 ttl=64 time=19.4 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=11 ttl=64 time=11.6 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=12 ttl=64 time=19.4 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=73 ttl=64 time=10.9 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=74 ttl=64 time=7.81 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=75 ttl=64 time=31.9 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=662 ttl=64 time=35976 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=698 ttl=64 time=65.5 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=699 ttl=64 time=34.5 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=700 ttl=64 time=41.6 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=701 ttl=64 time=28.7 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=702 ttl=64 time=50.3 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=703 ttl=64 time=26.8 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=706 ttl=64 time=48.7 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=707 ttl=64 time=21.5 ms
64 bytes from aaaa::212:7401:1:101: icmp_seq=708 ttl=64 time=35.0 ms

```

Figure 3-20 Terminal window within Contiki wherein the server IPv6 address is pinged to ascertain successful establishment of connection with the server.

Using 6LoWPAN analyzer with PCAP, details pertaining to the 6LoWPAN packets that are being transmitted by the nodes can be viewed, as shown in figure 3-21.

No.	Time	From	To	Data
1	01:12.596	4	3	97: 15. 4 D 0012:7404:0004:0404 FFFF IPHC ICMPv6 RPL DIO 11110011 00000000 00000000 00001100 040E0008 0C0A0700 0100...
2431	02:00.020	1	2	97: 15. 4 D 0012:7401:0001:0101 FFFF IPHC ICMPv6 RPL DIO 11110011 00000000 00000000 00001100 040E0008 0C0A0700 0100...
34	02:01.780	3	2, 4	97: 15. 4 D 0012:7403:0003:0303 FFFF IPHC ICMPv6 RPL DIO 11110011 00000000 00000000 00001100 040E0008 0C0A0700 0100...
35	02:04.055	2	[3 d]	76: 15. 4 D 0012:7402:0002:0202 0012:7401:0001:0101 IPHC ICMPv6 RPL DAO 1E4000F5 11110011 00000000 00000000 0000110...
36	02:05.846	4	3	76: 15. 4 D 0012:7404:0004:0404 0012:7403:0003:0303 IPHC ICMPv6 RPL DAO 1E4000F5 11110011 00000000 00000000 0000110...
37	02:05.859	3	2, 4	76: 15. 4 D 0012:7403:0003:0303 0012:7402:0002:0202 IPHC ICMPv6 RPL DAO 1E4000F5 11110011 00000000 00000000 0000110...
38	02:05.873	2	[3 d]	76: 15. 4 D 0012:7402:0002:0202 0012:7401:0001:0101 IPHC ICMPv6 RPL DAO 1E4000F5 11110011 00000000 00000000 0000110...
39	02:14.110	5	2	97: 15. 4 D 0012:7405:0005:0505 FFFF IPHC ICMPv6 RPL DIO 11110011 00000000 00000000 00001100 040E0008 0C0A0700 0100...

```

IEEE 802.15.4 DATA II
From ABCD/0012:7403:0003:0303 to ABCD/FFFF
IPHC HC-06
tf = 3 nhc = false hlim = 2 cid = 0 sac = 0 sam = 3 MCast = 1 dac = 0 dam = 3
IPv6 ICMPv6 TC = 0 FL: 0
From FE80:0000:0000:0000:0012:7403:0003:0303 to FF02:0000:0000:0000:0000:0000:0000:001A
Type: RPL Code: DIO
InstanceID: 30 Version: 240 Rank:1226 MOP: 2 DTSN: -11

Payload (64 bytes)
11110011 00000000 00000000 00001100 040E0008 .....
0C0A0700 01000001 00FFFFFF 081E4040 00000000 .....@.....
00000000 00000000 AAAA0000 00000000 00000000 .....
00000000 .....

```

Figure 3-21 6LoWPAN analyser within Contiki wherein details associated with the 6LoWPAN packets being transmitted by the nodes can be viewed.

Upon pasting the IP address of the server within Mozilla Firefox web browser, information pertaining to the only neighbour (Node ID 2) and all the routes can be viewed (on the local web), as shown in figure 3-22.

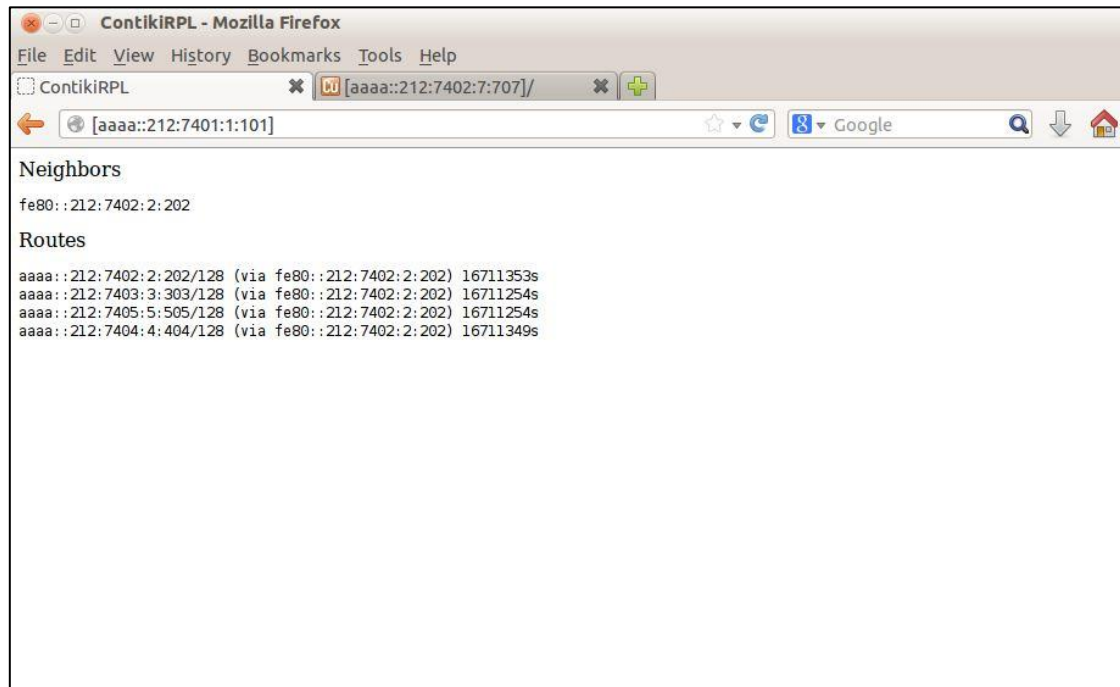


Figure 3-22 Upon entering the server address within the Mozilla Firefox web browser, details pertaining to neighbours and routes can be viewed.

3.7.5 Scalability

Another salient feature offered by Cooja which renders it desirable for this research work is the aspect of scalability i.e., its ability to test the behaviour of algorithms and protocols on scaled-up networks [119].

3.7.6 Limitations or Challenges Associated with Cooja as a Virtualization Platform

Although Cooja arguably serves as a viable (open-source and readily available) tool for WSN virtualization and is equipped with quite a few desirable capabilities and features, certain real-life factors restrict its scope as an infallible virtualization tool. For example, the radio signal strength or 'RSSI' signal value between two physical nodes deployed in an environment is difficult to model, which cannot be represented accurately within a Cooja-based virtual environment. Cooja has an in-built model that computes the RSSI of a virtual node with respect to another node. The variations in RSSI between the two nodes can be clearly observed when a particular node is moved apart from the neighbouring node. However, these variations are merely distance dependent and does not consider dynamics in the environment. Real-life factors (pertaining to the physical environment) affecting RSSI do not figure within the virtual

Cooja environment (since they cannot be catered for i.e., modelled within it). Attempting to inaccurately model these environmental factors within the virtual environment of Cooja may inevitably tend to introduction of undesired inaccuracies (implicating on any focussed performance measures viz., packet loss, latency, etc.) rendering it ineffective as a virtualization tool. As a crude optional workaround around this deficiency, random sensor values could be generated using the random function ‘rand()’ available in the C program. However, this tends to inevitably result in inaccurate virtualization outcomes. Feeding of such real world sensed data onto Cooja for simulation purposes is therefore an important consideration to be taken into account whilst ensuing upon testing and determining the best re-orchestrations to be applied to before implementation.

3.8 Conclusion

This chapter provides an insight into the research methods as well as the tools used for this research work. The viability of employing the open-source OS (Operating Systems) of Contiki and Integrated Development Environment (IDE) as a tool for both software development and for sensor network modelling are discussed. It then documents information pertaining to the Contiki-ported hardware platform of the Contiki-ported Texas instruments-based CC2538 as well as Contiki’s in-built network virtualization platform of Cooja. The necessary justification towards choosing them for implementing the physical and virtual sensor network environments (respectively), of our proposed architectural organization has also been provided.

While certain salient features of these tools have been put forth by means of example cases of implementations, some of the challenges associated with these tools have also been brought to the fore. Besides, the significance of the database developed for storing and visualizing of the sensed data has been discussed.

Chapter 4

Software-Defined WSN Concept Development

4.1 Introduction

This chapter offers insight into the concept developed towards designing an approach for ‘Software Defined Wireless Sensor Networks’ (SDWSN). Broadly speaking, the two elements of ‘novelty’ of this research work pertain to a) flexible ‘re-orchestration’ of sensor network functions via modularization and software-controlled virtualization of the same, and b) determining the latency or downtime associated with such dynamic re-orchestration. For the former, emphasis has been laid on the identification and consolidation based on the definitions of the core functional components integral to any Wireless Sensor Network (WSN) formation. It also included outlining certain intrinsic operational parameters that can be subjected to software manipulation. This paves the way for modularization of the firmware with which a WSN node is configured, thereby allowing for virtualization of the same.

A simplistic yet viable modular approach for flexible WSN functional re-orchestration that has been adopted here is ‘Reconfigurability’. An integrated or unified firmware, accommodating all the necessary functional modules is loaded onto the hardware nodes, depending on their ability to accommodate for them. The constituent firmware ‘modules’ could then either be invoked and modified individually, or in combination of one another. These interactions occur in a dynamic manner by means of external messages acting as commands to realize the desired re-orchestrations. The proposed architectural solution for SDWSNs takes into cognizance the potential offered by the cloud towards planning of the necessary re-orchestrations at the virtual level. By means of allowing for running soft-trials of re-orchestration scenarios, cloud-provisioned virtualization forms an important component of re-orchestration mechanism in regard to a) obviating the requirement for hardware for testing purposes, b) expediting derivation of the re-orchestrations to be applied onto the physical WSN and c) enabling foreseeing of the implications of doing so to some extent. The latter sections of this chapter revolve around the re-orchestration process itself and the ‘network downtime’ associated with the same. For this purpose, at the foremost, the viable strategical phases a network is deemed to go through from the instant of detection of an event necessitating re-orchestration to the subsequent ‘planning’ for the same and the resumption of normal ‘service’ or dataflow post re-orchestration implementation, is outlined. Secondly, formulation of a generic model for the latency entailed solely by re-orchestration process is ensued upon. This is followed by the chapter’s conclusion.

4.2 WSN Softwarization: Key Pre-requisites

The term ‘Softwarization’ has been referred to as utilization of software-based solutions, as opposed to proprietary, dedicated hardware-based solutions to offer network service solutions [120]. In the context of WSN, it can be interpreted as a paradigm that lends itself towards enabling running of a certain WSN function in software, rather than on conventional low power

hardware devices [121]. By means of allowing for functions to be manipulated via software control, the principle advantages offered by softwarization include increased flexible configurability, time-efficient implementation and operation, low maintenance and management costs, etc., coupled with remote configurability [122-123].

Establishment of a SDWSN organization that is capable of undergoing desired flexible re-orchestrations in a seamless manner necessitates logical consideration in regard to certain essential pre-requisites it may entail. From a formalization standpoint, identification of the core WSN functionalities viz., Leaf, Router and Gateway functionalities, is deemed to be the first pre-requisite herein. Clear definition of the aforementioned three essential functionalities so identified forms the second pre-requisite that paves the way for their modularization. Ought to be regarded as a major component of (any) flexible organization, modularization tends to form the third pre-requisite within our approach. Configuring a sensor device with modular firmware comprising of requisite functional modules could allow for seamless invocation of any ‘reusable’ logical software module present within the firmware used for configuring a given WSN node. Creation of library of functions within the firmware employed could also contribute towards augmenting the degrees of freedom encompassed by the individual ‘network functions’, and thereby the overall system. However, this necessitates identifying and clearly discerning the ‘core’ roles that could be attributed to each of the elementary functions that form the ‘building blocks’ of any WSN organization in the underlying Physical layer. Besides catering for reusability of software functions and thus, overall flexible customizability, modularity opens the door for their virtualization and conduction of pertinent soft trials within the cloud. Such virtualization-aided ‘planning’ of the re-orchestrations to be realized within the physical layer could offer itself towards significantly expediting the overall process (of network re-orchestration) and hence deserves emphasis as a key component within SDWSN organizations. These are conceived to be important (pre-requisites) from the flexibility standpoint. The aforesaid (organizational) elements are deemed to be critical pre-requisites that would enable the sensor network organization to smoothly undergo the process of software-defined re-orchestration.

4.3 Key Modular Components for Flexible WSN

Any WSN organization is perceived to be composed of devices statically configured with either of the three core functionalities, namely the Sensing, Routing and Gateway functions [25-26]. Each of these standalone functions ought to be treated as a ‘software module’, which when used to (pre-) configure a given physical sensor-transceiver, defines its functional role within the overall network process or operation. Based on their respective classical definitions, each of these primary core functions tend to be characterized by a set of certain operational components or tasks which they are meant to execute. For example, the key operational components of a physical sensor-transceiver node configured with the leaf functional module include Sensing and Data Acquisition, Data Management and Computation and Communication (with sink node). Similarly, a node configured with ‘router’ functional module comprises of the basic operational components of Data Reception and Storage, requisite MAC Layer Manipulation(s) and Data Forwarding or routing data packets to either other leaf nodes or other routers or to the higher layer of the Gateway. Lastly, a device configured with the Gateway

functional module discharges the core responsibilities of acting as a sink for the physical data received from all its constituent router nodes. The system then performs the requisite protocol conversion (rendering it suitable for transmission or communication over the Internet) and finally transmits the data over the Internet i.e., serving as an access point to the cloud over the Internet. All the possible tasks pertaining to each of the identified core WSN functions are as outlined within the following subsections, along with their associated descriptions.

4.3.1 WSN Leaf Function

The operational role executed by Physical sensor transceivers configured with the ‘leaf function’ can broadly be classified into three tasks viz., ‘Sensing and Data-Acquisition’, ‘Data-Management and Computation’, and finally communication of the sensed data to a sink node, as depicted in figure 4-1. Each of these tasks entails certain node-operational parameters that can be presided over by software control. Commonly referred to as end devices, the leaf nodes typically tend to have limited battery power.

Sensing and Data Acquisition

The first and foremost stage of the Leaf functionality pertains to Sensing and Data Acquisition. This level entails the two node-operational parameters of sensor selection and data acquisition rate(s), as depicted in figure 4-1.

WSN nodes may be mounted with multiple sensors allowing for sensing heterogeneity. Each such sensor has a particular sensing function associated with it within the leaf firmware employed for configuring the nodes. Such leaf-firmware sensor nodes could be configured to acquire data from either one or all of its sensors simultaneously. Prevailing sensing requirements may dictate the criteria for selection of the sensors available at the disposal of the hardware employed within the WSN, including configuring the sampling rate for each of the sensors (on an individual basis) within the firmware.

Data Management and Computation

The data sensed by the leaf node needs to be stored and preferably subjected to a certain degree of processing prior to transmission (to a sink node). These two sub-tasks constitute the operational task of Data Management and Computation wherein data samples are sensed by the node and buffered (typically within an array of a certain size defined within the firmware used for configuring the node). The size of such an array for data-buffering purposes could be increased or decreased via exerting software control over the firmware. The extent up to which the buffer size may be increased via software control would be dependent on (or constrained by) the hardware employed as the sensor-transceiver. A separate array could be defined for each sensor variable, in case the leaf node possesses multiple sensing capabilities (i.e., sensing heterogeneity, as discussed above).

Secondly, the data so stored could then be subjected to certain computation algorithms such as averaging of, say, every ten samples so as to generate and schedule a single averaged value to be transmitted as opposed to all the ten sample values. Alternatively, a query processor

may be included as part of leaf node's firmware in the form of a conditional, allowing for scheduling of transmission of only those data samples which meet the condition specified within the 'conditional' (e.g., allowing only those data samples to be scheduled for transmission that are equal to or greater than a certain set threshold, and filtering the rest). By effectively reducing the number of samples to be transmitted, such data management techniques can reduce the communication overhead a leaf node may be subjected to, thereby contributing towards battery power conservation and prolonging node lifetime.

Data Communication

Advancements made in the field of embedded SoC systems have enabled development of WSN hardware nodes that could be configured to operate on different channels. Each type of hardware may have a range of frequency channels with which it could be configured. The desired channel needs to be specified in the pertinent section of the firmware (within the appropriate header or configuration file) during firmware configuration or update process. This tends to open the door for frequency-based clustering (i.e., multi-channel assignment) within a given WSN, allowing for parallel transmission of data, and could therefore be resorted to as solution towards alleviating interference in dense WSN deployments.

The RF (Radio Frequency) transmission power output of a node refers to the power that is produced by it at its output during transmission [124]. It is typically specified in 'dBm'. Exertion of software control over the output power of radio transmission of the constituent nodes tends forms an essential component in WSN operations towards either overcoming poor signal quality issues (that may arise owing to physical and/or environmental factors viz., obstructions, adverse weather conditions, etc. or long distances) amongst nodes or simply for the sake of increased reliability (via ensuring better signal quality reception, i.e., RSSI at the receiving node) of radio communication amongst them.

It is however worthwhile to keep in mind that the process of communication or transmission of radio message data by a node to another node tends to entail a significant amount of battery power. Higher the radio transmission power, greater is the current consumption of the node (resulting in decreased battery lifetime, more so if the node transmits data frequently) as well as increased interference [125].

Dynamic manipulation of the radio transmission power output of the node through software control is therefore of significant importance towards striking a trade-off between the battery power expended (given the limitations in regard to the battery power available to the node) and the reliability of communication amongst the nodes (that may be deployed in harsh environments or spaces accounting for low RF penetration or at distances faraway from each other)[126].

Data Communication rate of a WSN node (, say, a node configured with a leaf function) refers to the reporting interval of the data sensed by it to another node (say a node configured with the router function). Configuration of this node-operational parameter within the firmware employed is critical as regards to its battery lifespan, contention of the wireless medium being utilized for transmissions, etc. [127]. While nodes may be configured typically with 'fixed' data communication rates, it is important to perform software manipulation upon the same (in

a dynamic manner) so as to ensure that significant dynamics pertaining to the external monitored phenomenon are not lost.

The data sensed (or received) and stored by a WSN node is transmitted to another node by means of the requisite transmission function. The communication usually takes place through the radio module [128] and is in accordance with the configuration with respect to the aforementioned node operational parameters within the firmware.

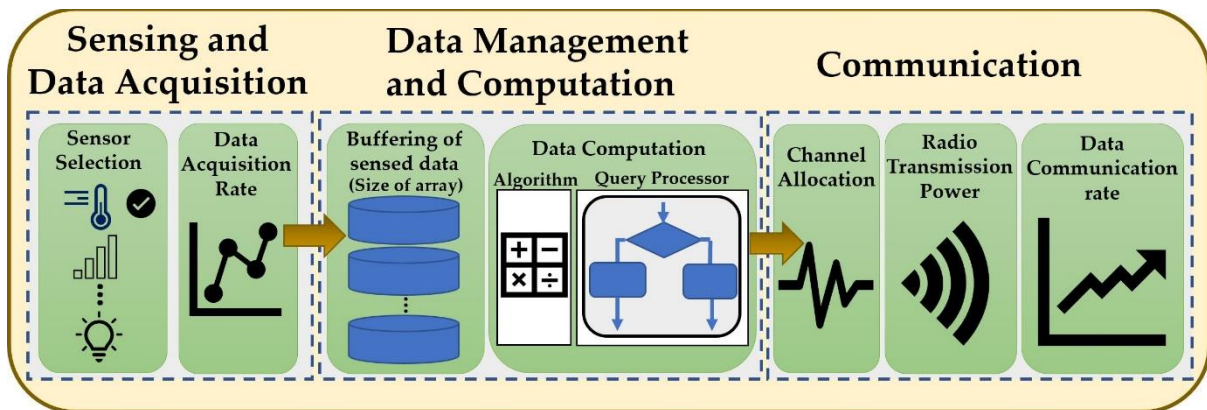


Figure 4-1 Operational tasks and associated components pertaining to Leaf function.

As a means to put the above in the context of a real-world example, consider the case of forest fire monitoring and detection via WSNs. As discussed in the introductory chapter, forest fires may occur as a manmade hazard or as a natural climatological phenomenon [129-131], uncontrolled spread of which could result in widespread incineration of forest vegetation and destruction of animal habitat. Besides, it may prove to be detrimental to the surrounding environment (via worsening the quality of air and posing increased health risks to all nearby wildlife). Monitoring of forest areas through deployment of WSNs tend to provide a viable avenue [132-136] towards not only detecting a sudden fire outbreak and localizing its epicentre but also prevent it from spreading to neighbouring areas, where possible. Flexile functioning of the WSN so deployed however, forms a desirable operational requirement, so as to effectively capturing the dynamics associated with fire outbreak and track its spread. For example, if a fire were to suddenly erupt in a certain area of a forest region monitored by a dynamically flexible WSN, the leaf-function-configured sensor nodes deployed closest to the localized area engulfed by the fire (in its initial stages) could firstly respond via undergoing software-defined re-orchestration to select or enable the desired sensor(s) viz., temperature, humidity, precipitation and windspeed [137-144]. Secondly, the respective data acquisition rate(s) could be increased (for capturing the significant dynamics associated with occurrence of such an event in a more accurate manner) whilst amplifying the internal buffering capacity (to accommodate for the copious amount of sensed data). Thirdly, the sensed data so acquired could be subjected to data computation for data compression purposes, if required. Finally, the data could be transmitted to the governing router node at a higher (than normal) data communication rate as well as at an increased radio transmission power output (, if required). While these dynamic node-operational re-orchestrations occur simultaneously across a certain number of nodes in close proximity to

the region lit by the fire, the WSN must also be capable of responsively re-orchestrating its operational behaviour from the overall network standpoint (e.g., resorting to contention-based channel-access method, if configured to operate in accordance with a schedule-based channel-access method under normal circumstances), including that from a topological perspective, especially during times of network fragmentation events caused during such calamities. The former aspect (MAC-based channel access method manipulation) is discussed in sub-section 4.3.2 while the latter (topological re-orchestration and ensuing network performance improvement) is elaborated upon via examples in section 4.4.2.

4.3.2 WSN Routing Function

In WSNs, sensor nodes configured with the routing function execute the role of relaying the sensed data from the ‘leaf’ nodes to either other routers or the Gateway and vice-versa, via the most suitable route, or in accordance with the routing protocol employed (e.g., mesh protocol) as well as playing the role of a cluster-head (and thereby as a temporary sink) for its constituent group of leaf nodes. The routing function firmware is characterized by the three tasks of ‘Data Reception and Storage’ (of sensed data received from the leaf nodes within its cluster), ‘MAC layer manipulation’ and ‘Communication’ as depicted in figure 4-2.

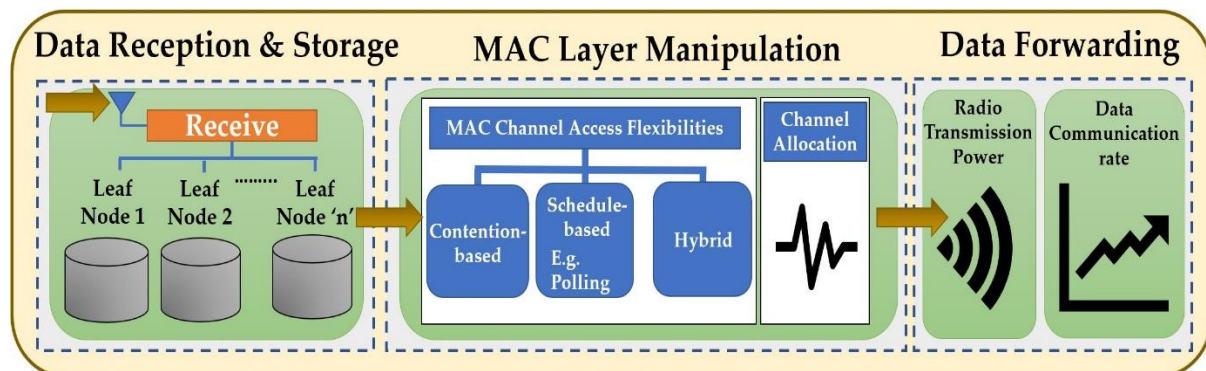


Figure 4-2 Operational components and associated tasks pertaining to ‘Router’ function’.

As part of its role of acting as a ‘cluster head’ for the constituent leaf nodes present within its cluster, the router node firstly accommodates for the data transmitted by them (i.e., the constituent leaf nodes). By means of provisioning for multiple buffer arrays. As mentioned earlier, the sizes of these buffer arrays can be adjusted as per requirement via software control but within the constraints imposed by hardware limitations.

Software manipulations of the MAC layer protocols could be ensued as a means to exert control over the channel access methods whilst multiple nodes share the same wireless transmission medium to communicate their data.

Radio communication operations accounts for a major part of a WSN node’s energy consumption [145], thus necessitating conscientious management of the radio. Moreover, data collisions may significantly impact radio communication operations taking place within dense WSN deployments, especially when a large quantity of the constituent nodes share the same

channel for data communication at the same time. MAC layer protocols provision for channel access mechanisms and can be presided over through relevant software control as a means to regulate the utilization of the shared communication medium. Two of such channel access methods are referred to as ‘Contention-based’ and ‘Schedule-based’ ‘channel access methods. Implementation of ‘Scheduled-based’ or ‘fixed assignment’ channel access methods such as ‘Polling’ results in each node transmitting its data only when polled[146], virtually resulting in ‘collision-free’ communication. This channel access method however may not be suitable for service requirements (involving rapidly changing significant dynamics within the monitored external phenomenon) necessitating the relevant nodes to communicate data at a higher-than-normal rate sans predetermined time slot (i.e., contention-based channel access methods).

The Data Forwarding task entails the two operational parameters of Radio Transmission Power Output and Data Communication Rate. As explained earlier within section 4.3.1, these operational parameters refer to signal strength and rate of transmission of data (to another sink node i.e., another router node or gateway node) and can be flexibly manipulated to suit service requirements in real-time.

4.3.3 WSN Gateway Function

The Gateway unit acts as a local ‘sink’ for the entire incoming sensed data transmitted by the connected leaf nodes and router nodes and caters for the requisite protocol conversion and relay the sensed data to the ‘remote cloud-server’ over the Internet[41-42, 50] using the transport layer. The three core tasks encompassed by the Gateway function, namely, ‘Acting as a Sink Node’ (entailing reception and storage of data relayed by the router node), ‘Protocol Conversion’ and ‘Pushing of Data to Remote Server’ over the Internet, are as depicted in figure 4-3.

The data relayed by the router node is accommodated for within the gateway by means of one or multiple buffer arrays of certain sizes (declared within the firmware and dependent on both number of types of incoming sensed variables, as well as the total number of leaf nodes). This data then undergoes the requisite protocol conversion. For this we extracted the data from the packet received on the gateway and created a new packet with destination and source IP added. This enabled data to be sent to and received from the virtual layer hosted on the server. For pushing the data to the ‘remote-server’ we used the ‘GET’ command of the REST API.

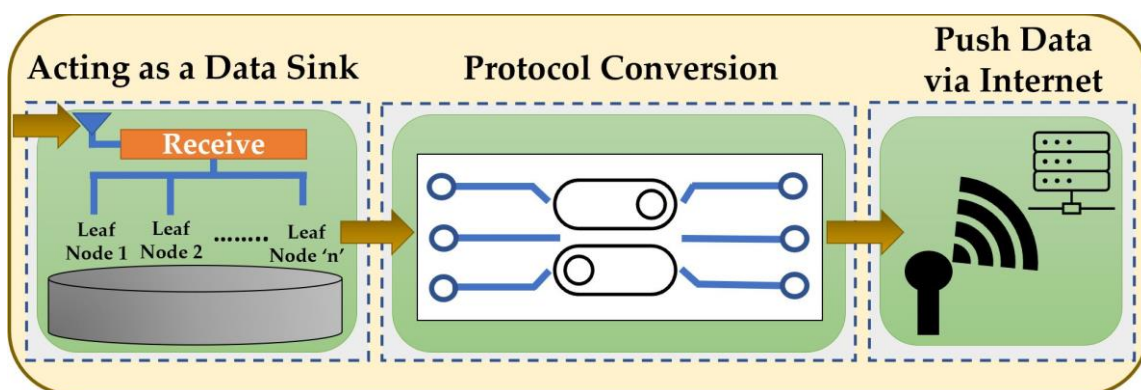


Figure 4-3 Operational components and associated tasks pertaining to ‘Gateway’ function.

Owing to the availability of relatively high computation power and memory space possessed by these units, certain other core functions (somewhat tantamount to edge computing) prior to transmission over to the cloud, could also be incorporated as ‘auxiliary functions’, if required.

In the context of the forest fire monitoring example, data pushed by the Gateway node over the Internet to a remote cloud server could be used for continual monitoring of the key sensor parameters pertaining to forest fire monitoring viz., temperature, humidity, precipitation and windspeed [136-143] under normal circumstances and for triggering an alert to notify firefighting rescue team stationed closest to the area in the event of a sudden fire outbreak.

4.4 WSN Re-orchestration

In order to adequately cope with the wide variety of dynamic service requirements, a WSN organization ought to be able to tap into any and all of such aforementioned operational flexibilities (outlined in section 4.3) in a responsive manner. Software-defined flexible WSN operation offers itself as a viable solution in this regard since it renders the system capable of undergoing flexible re-orchestration at both the node and network levels in a dynamic manner.

Node-level re-orchestrations refer to manipulation of the intrinsic node-operational parameters (of an individual WSN node) through software control. The ‘functional role’ executed by the node being unaffected, such re-orchestrations do not influence the topological orientation of the network. Certain of such node-operational parameters that could be subjected to software manipulation include ‘Sensor Selection’, ‘Data Acquisition’, ‘Buffering of Sensed Data’, ‘Data Computation’, ‘Channel Allocation’, ‘Radio Transmission Power’ and ‘Data Communication Rate’, as elaborated in section 4.3.1 (as well as section 5.4 in chapter 5).

Re-orchestrations at the network-level, on the other hand, may either impact the nature of the dataflow transpiring within a WSN (e.g., defining the ‘channel access method’ at the MAC layer through software) or result in alteration of the topological orientation of a given WSN (albeit, the change may be restricted merely to that of the functional ‘role’ of an individual constituent node). Although such re-orchestrations may not invariably result in a different topological formation altogether, they tend to offer a viable avenue towards converging upon desired topological orientations (albeit, possibly in an iterative way). This section attempts to elucidate the above assertions by means of a simple example wherein network topology and/or the flow of data within the network gets altered as a result of subjecting a couple of constituent nodes to such network-level re-orchestrations. Another example focusses on the implication of such topological re-orchestration (caused as a result of such network-level re-orchestrations) on the performance of a relatively bigger network.

4.4.1 Simple Network Re-orchestration: Topological Flexibility

Through an incremental example case, network re-orchestration(s) brought about by requisite software control, resulting in alteration of the topological orientation of a three-node WSN setup has been demonstrated herein. The network shown in figure 4-4a [26] below is initially configured to operate as a multi-hop WSN network wherein data sensed by the leaf node i.e., node 1 reaches node 3 i.e., the sink node (configured with the gateway function) via the intermediate node i.e., node 2 (configured with the router function).

Dynamic re-orchestration of node 2 (via software control), effectuating it to execute the role of a leaf node (as opposed to its original functional role of a router node) ceases the existing multi-hop operation offered by the network. Furthermore, invocation of the ‘Polling’ channel access method for all three of the nodes (resulting in nodes ‘1’ and ‘2’ communicating their data to node 3 only when polled by node 3) at the same time instant transforms the same network to act as star topology-based network, as depicted in figure 4-4 b [26].

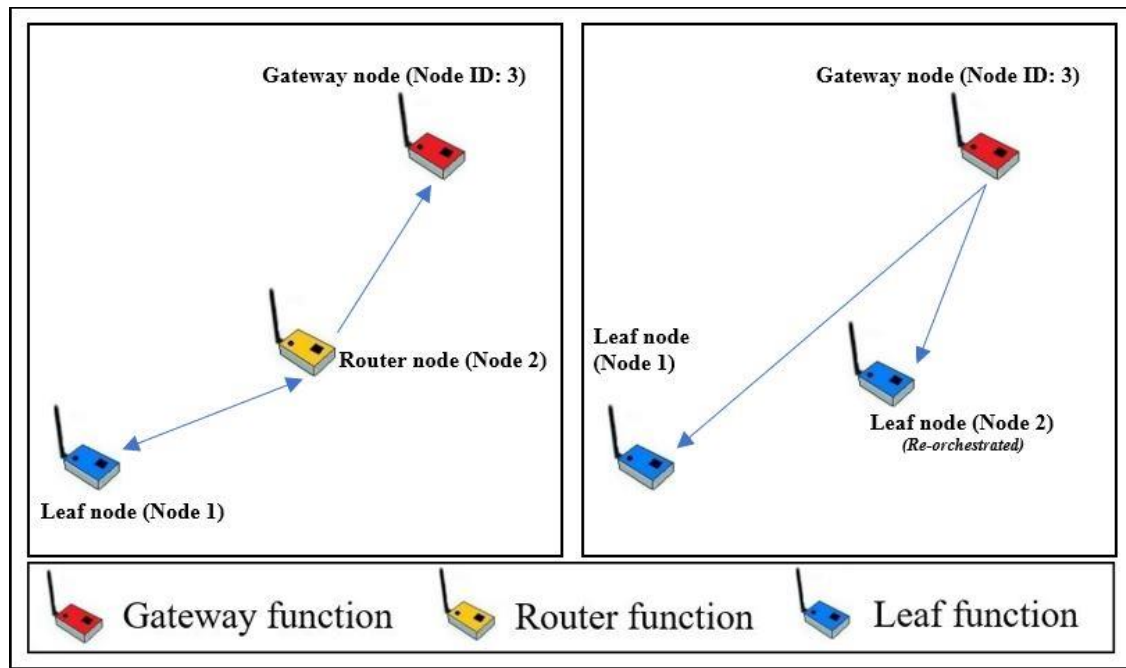


Figure 4-4 Example topological re-orchestration from (a) multi-hop network to (b) star network brought about by simple network manipulation.

This simple example highlights the impact that such minor re-orchestrations may have towards altering the topological orientation of a given WSN, whilst reflecting the saliency of exertion of software control for the same.

4.4.2 Example Network Topological Re-Orchestration Towards Improved Network Performance

Software defined topological re-orchestration of sensor networks is essential towards

- as an operational mode contributing to adaptive reconfiguration of the sensor network as it interacts with the monitored physical phenomena (in order to deliver better network performance)
- re-orchestration in the event of network fragmentation or a sensor network service to an end-user

The following example deals with an example simulation of a case to analyse the improvement in network performance as a result of sensor network topological re-orchestration. Herein, a 9-node network, consisting of 7 routers represented by the yellow-coloured nodes (capable of turning into leaf nodes), one leaf node (represented by the blue-

coloured node) and one gateway node (represented by the red-coloured node), thereby offering multi-hop topological behaviour, is considered. As shown in figure 4-5 a [26], nodes ‘2’ to ‘8’ within this network forward the data obtained from ‘node 1’ over to ‘node 9’ in a uni-directional fashion. Such an arrangement could not only result in relatively greater amount of delay in traversal of the data sensed by the only end node i.e., node 1 to the ‘sink node’ i.e., node 9 but also packet loss, to certain extent.

Upon subjecting the ‘router-configured’ nodes, except node 7 (as depicted in figure 4-5 b [26]) to ‘software control-triggered’ re-orchestration, directing them to execute the role of leaf nodes and communicate their data to node 7, the network ceases to be a multi-hop network and begins to operate as ‘star-topology’ based network. This topological arrangement offers the advantage of faster data communication within the network (owing to single-hop communication from the all the leaf nodes to the router node), mitigating the loss of packets during the same, whilst accommodating for greater sensing coverage area.

Dynamic re-orchestrations pertaining to the channel access methods (e.g., from contention-based channel access method to polling-based channel access method) could offer further mitigation (i.e., minimization) of packet losses during the ongoing network operation.

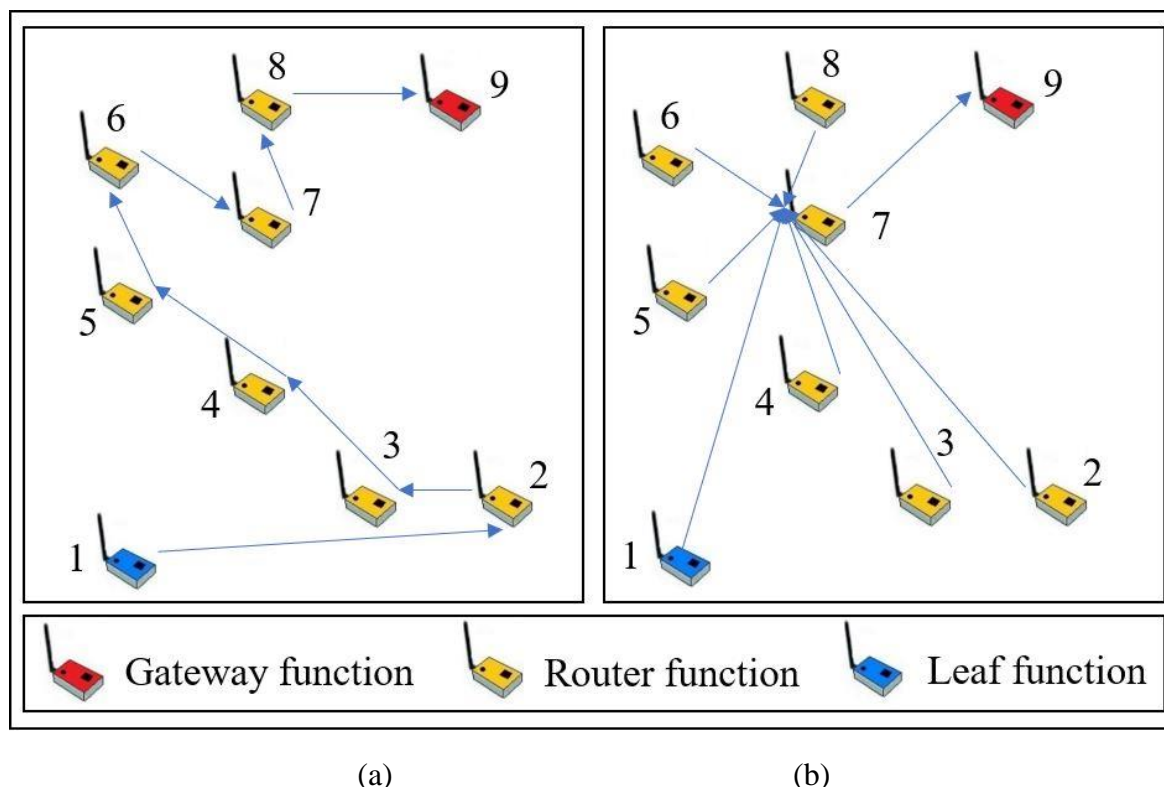


Figure 4-5 Example topological orientations a WSN can adaptively re-orchestrate to as a result of software-defined re-orchestration: (a) Multi-hop Topology and (b) Star Topology [26].

4.5 SDWSN Design Approach

Broadly speaking, both re-configurability and re-programmability-based approaches need to be looked at and adopted (as suited) in the pursuit of attaining considerable degree of desired dynamic and flexible SDWSN operational behaviour.

In regard to above account, it is deemed worthwhile to reiterate the argument advanced towards undermining the ability of hardware sensor–transceivers to switch to a different functional role or assumption of multiple functional roles is based on the generalized (and narrow) premise that sensors are rigidly characterized by highly integrated and specialized design, are bound by memory and resource-constraints and are not particularly well-suited for executing multiple tasks simultaneously.

This argument is, however, relative. Advancements made in the field of embedded (SoC) technology have ushered in resource-rich devices encompassing enhanced memory capacity and ample computational capability. Such progressions coupled with the modern trend of edge or fog computing enables either dynamic reformulation of the prevailing functional role being executed by the wireless chip or integration of multiple functionalities onto such a single wireless microcontroller chip.

Thus, based on the resourcefulness of the target hardware platform employed, either the prevailing i.e., existing elementary, modular standalone function could be replaced with another core functionality or two or more of the aforementioned elementary, modular standalone function could be conflated within the same hardware device (by means of software control), enabling multi-functional capability.

4.5.1 Re-configurability-based Approach: Realization via Unified Firmware

Command-driven Re-configurability-based approach, for example, could be realized via conditional execution of the clearly defined, logical software modules present within the code (with which a node is configured) i.e., via employing if-conditional statements or switch-case statements within the code). Depending upon the capability of hardware to accommodate for and execute multi-functional roles, the WSN nodes are configured with either a unified firmware i.e., one consisting of multiple necessary functional modules (viz., leaf and router, including additional modules) or with a firmware responsible for execution of any one functional role only (e.g., either leaf, router or gateway functions).

Such a unified firmware so loaded onto a particular node allows it to switch over to another distinct operational mode, the functional module of which is already defined within it. For example, if a WSN node is configured with a firmware comprising of leaf, router and gateway modules, it can re-orchestrated to execute either of such discrete, unique functional roles. Such flexible re-orchestration of the functional role executed by the resourceful WSN nodes could be brought about by means of an appropriate external command signal frame or message that is issued to it from the cloud layer (, as explained in section 4.4), resulting in execution of only the desired pre-loaded software modules present within the code, whilst disabling the inessential modules, during run-time. For example, re-orchestrating the functional role of a device from leaf-node to that of a router-node would involve disabling the operational modules pertaining uniquely to the leaf node (e.g., sensing function) as stated within its pseudo code put

forth in figure 4-6) and enabling the operational modules pertaining uniquely to the router node, by means of an external trigger signal message. Such allowance for assumption of desired configurational status is however subject to the vital assumption that the requisite distinct, core functional software modules of for example leaf and router are present within the code employed for configuring the FFD (i.e., Fully Functional Device) devices.

An example integrated firmware consisting of leaf and router modules is as depicted in figure 4-6 below.

Example of a unified firmware used for configuring a node that is capable of accommodating for and executing operational activities pertaining to both router and leaf functional roles

Initialization of arrays: Declare and initialize one or multiple arrays (if need be, depending upon the number of types of incoming sensed variables) of a certain size (depending upon the number of leaf nodes within its cluster) to receive all the incoming sensed data variables

If external radio message flag received directing execution of the leaf function sub-modules

Sensor selection: Execute the desired sensor function(s) (one or more) and acquire the sensor readings

If external radio message flag received directing execution of the router function sub-modules

Receive sensed data: Receive and store incoming sensed data from all constituent leaf nodes on a node-by-node basis

Buffering of data variables to be transmitted: Store received data within an array of requisite size

If external radio message flag received directing execution of the 'leaf function' sub-modules

Channel Allocation: Set the node to communicate using (one of the available and) desired channel(s)

Buffering of sensed data: Store the sensed data within an array of requisite size along with node ID

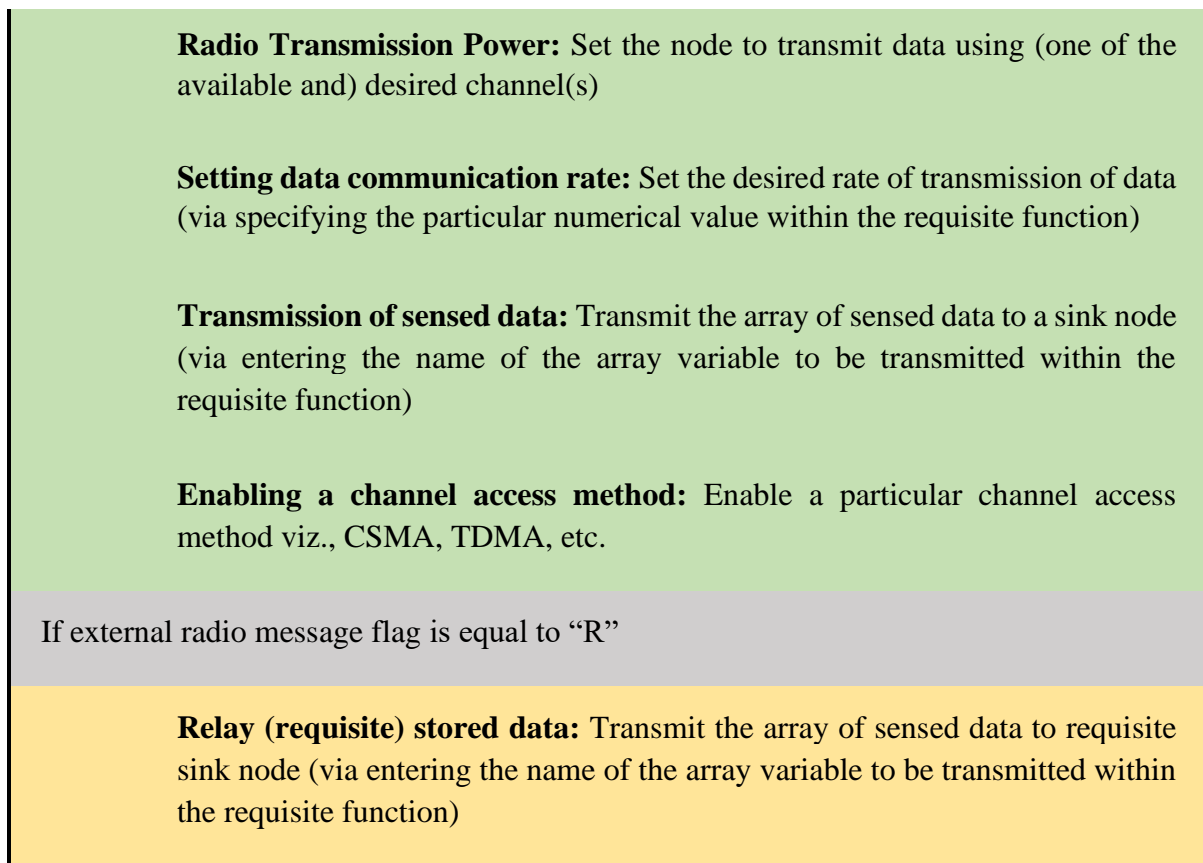


Figure 4-6 Example unified firmware consisting of operational components pertaining to both Leaf and Router functions.

Herein, operational activities highlighted in yellow refer to leaf functional modules whereas those highlighted in ‘green’ pertain to router functional modules.

As alluded to previously in chapter 2, adoption of the Re-configurability-based approach allows for WSN organizations to undergo re-orchestrations in a more flexible, reliable, and swift manner as compared to *Re-programmability-based* approach. Re-programmability involves transportation of part(s) or complete section of the program to the required node over the air. Such transportation tends to entail number of hops through which the data packet (containing the complete or part of the code) may need to traverse, making it susceptible to packet loss as well as increased delay (latency). Moreover, the problem of the node going into state wherein it may tend to reset continuously, potentially also gets alleviated [37].

It is worthy to concede that the above approach that involves presiding over a unified firmware using software controls may account for certain drawbacks. These include requirement of a microcontroller with a large flash memory space as well as possible inefficient utilization of the program memory space (in case certain of the pre-defined operational components within the unified firmware were to be rarely invoked). However, the same approach caters for WSN re-orchestration in a rapid fashion, rendering it better equipped to cope with the dynamic service requirements. It also eliminates certain challenges associated with *re-programmability-based* approaches including delay, unreliability of the wireless medium (it being prone to packet collisions, interference, etc.) leading to loss of (portions) of program codes during transmission (esp. when multiple nodes scattered over the deployed

region are to be re-orchestrated) and overhead (caused during writing of the program codes onto the flash memory) [147]. The justification behind considering forest fire detection and monitoring for the case study portrayed in chapter 6 of this thesis is to merely highlight the saliency offered by the ‘re-configurability’ approach towards ensuring that a WSN experiences as low packet loss and network delay as possible (as compared to other approaches such as ‘Re-programmability, OTAP, etc.) whilst undergoing re-orchestration. This approach could also be regarded as more viable when the volume of data flowing across the network is high (without having to flexibly manipulate certain elements the design of the WSN e.g., buffer size, delay tolerance capability, etc. towards equipping the system to cope with the high-load conditions, unless necessary).

4.5.2 Library of Software Modules: Towards Incorporation of Additional Functions

As a means to flexibly incorporate additional (hitherto-undefined) WSN functions, a library of ‘reusable firmware modules’ could be created and integrated within the firmware, as illustrated by the generic block diagram structure depicted in figure 4-7. Although the physical nodes would require manual firmware reconfiguration as a means to execute the newly acquired WSN functions, the frequency of undertaking such processes could be largely limited owing to the advantages offered by virtualization technology towards testing various node and network level re-orchestration scenarios beforehand. The following sub-section (sub-section 4.5.3) offers information on the saliency of virtualization towards abetting the proposed SDWSN design.

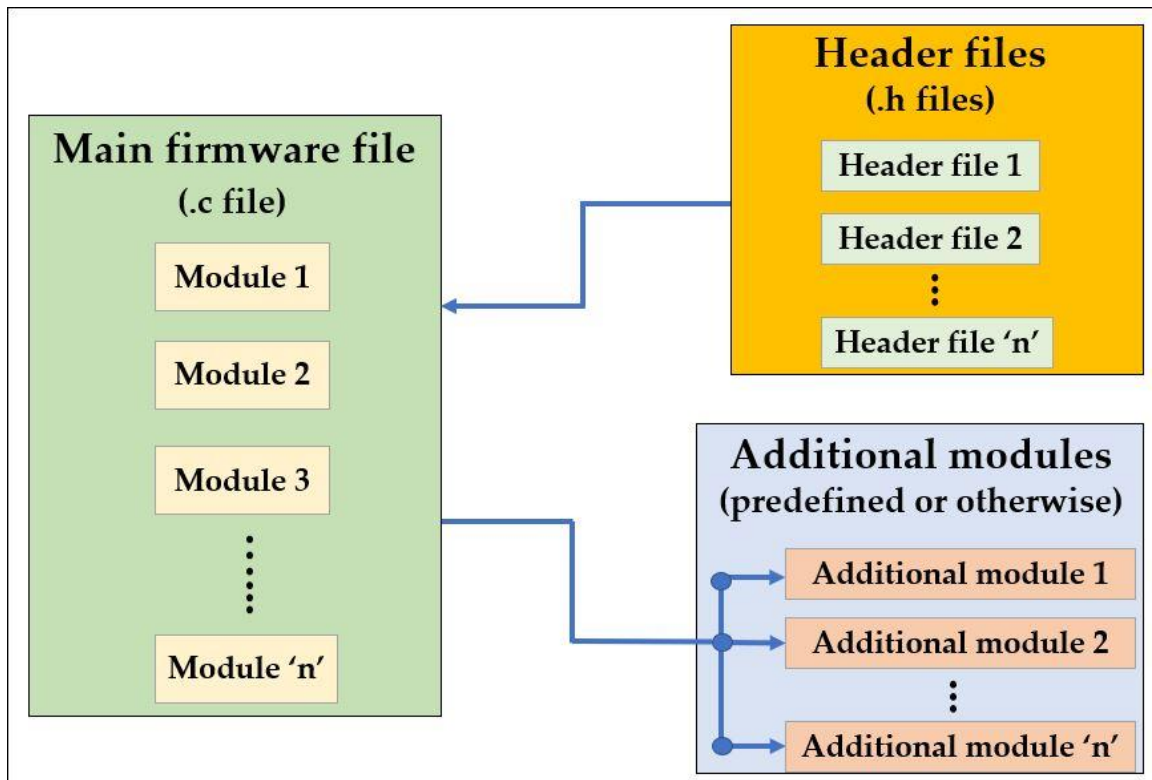


Figure 4-7 Generic firmware block diagram towards allowing for inculcation of additional functions with time.

4.5.3 Virtualization: Tool for Running and Testing Soft-Re-orchestration Trials

As alluded to earlier, virtualization refers to abstraction of the functions from the physical environment and ensuing upon their one-to-one representation in the virtual environment. Typically, this process involves decoupling of real-life physical functions by means of replicating the software program fed within the physical entities (physical sensor nodes) and running them over corresponding virtual entities (virtual sensor nodes), typically within a centrally located remote cloud server. Advancements accomplished within the technological domain of software application development have led towards development of certain Integrated Development Environments (IDE) that tend to not only allow for firmware development, compilation and configuration for physical nodes but also for their virtualization.

Depending on the process requirement, either a single node entity, a particular process (taking place in a certain portion of a network) or the entire network as a whole can be virtualized. While virtualization of a single node entity may merely entail creation of a virtual node configured with same 'firmware' (thereby catering for the exact logical behaviour), network virtualization involves creation of virtual model of the underlying physical network, reflecting the same (albeit typically proportional) structural or topological representation, etc.

This allows for mimicking the physical network. Moreover, owing to the reciprocity of the codes employed for both physical and virtual nodes (as facilitated by certain IDEs as mentioned earlier), the virtualization environment serves as a more viable avenue for testing various network re-orchestration scenarios (resulting from software-defined manipulations), as compared to a mere simulation environment. Although virtualization provisions for the aforementioned advantages, it may not directly be capable of catering for the replication of dynamics associated with the physical environment (for example real-world factors affecting the quality of the RSSI signal between two physical nodes may not be catered within the virtual environment).

4.6 Proposed System Architectural Organization

Figure 4-8 [25-26] shows the proposed cloud-based sensor network organization. It is composed of the cloud and physical environments. The stratagem herein consists of interactive collaboration between both the layers towards facilitating for the necessary identification, planning and execution of the re-orchestration process as necessitated by prevailing service requirements.

4.6.1 Physical Layer

The Physical layer accommodates for the underlying physical IoT-based sensor network wherein each of the core constituent standalone functionalities are entrusted with the responsibility to execute a unique role within the overall network operation. For example, the leaf function is responsible for sensing and acquisition of physical data whereas the Router node function executes the role of forwarding the sensed data to its desired destination within the network and/or to the Gateway. Finally, the IoT gateway function performs the task of transporting the sensed data to the 'remote cloud-server' over the Internet (in accordance with

the embedded IP protocol). Some of the hardware sensor-cum-hardware transceivers employed (such as the TI CC2538) are capable of being configured with multiple functionalities (i.e., they can perform the roles of both ‘leaf’ and router nodes simultaneously, if required). Moreover, they are also capable of sensing heterogeneous data. By virtue of one of its core roles as a ‘protocol converter’, the IoT Gateway unit serves as bridge between the physical and the cloud layers. The gateway units are typically resource-rich and computationally powerful, allowing them to execute ‘edge computing’-based tasks viz., compression, queuing, etc.

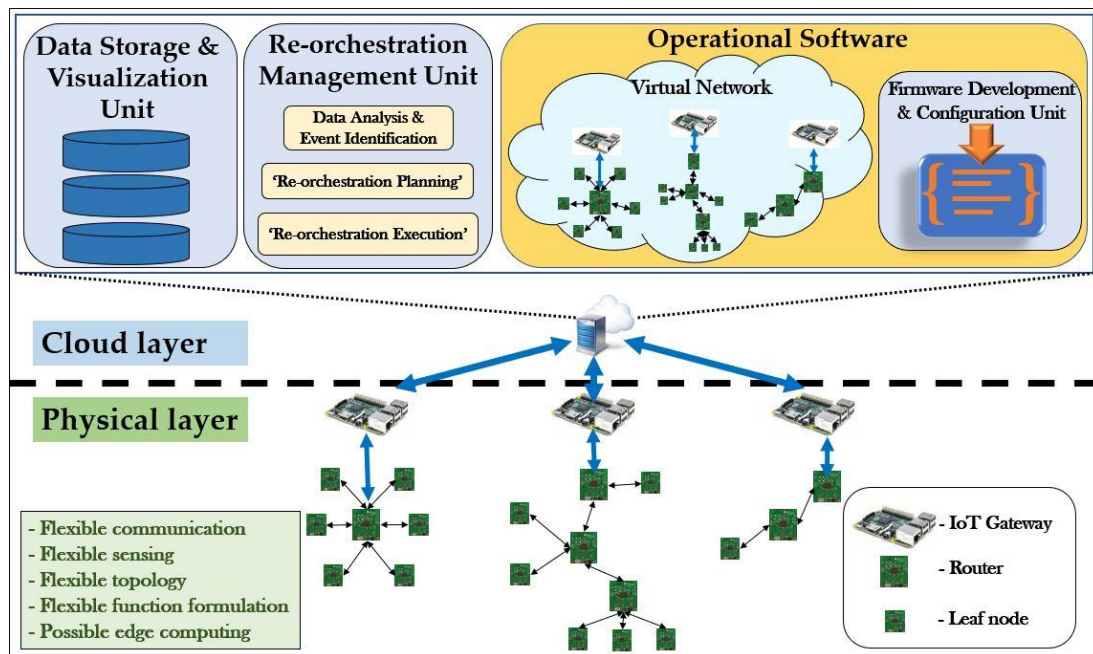


Figure 4-8 Proposed IoT-based sensor network organization [25-26].

4.6.2 Cloud Layer

As shown in figure 4-8 [25-26], the cloud layer encompasses three components for managing the flexible re-orchestration of the underlying physical layer, namely, ‘Data Storage Unit’, ‘Re-orchestration Management Unit’ and the ‘Operational Software Unit’. The sensed data received from the underlying physical layer is stored within the database offered by the ‘Data Storage and Visualization Unit’. This data is utilized for data visualization as well as both historical real-time data accessibility. The constituent virtualization and software resources prove to be instrumental not only towards working out suitable re-orchestrations via soft-trialling prior to actual physical re-orchestration but also foreseeing the implications of the same.

The ‘Operational Software’ refers to the Integrated Development Environment (IDE) platform that allows for development and implementation of the firmware to be employed for physical nodes. As alluded to earlier, certain modern-day IDEs also provision for virtualization i.e., creation of virtual nodes compiled with the exact same firmware as that employed for the corresponding hardware. By means of allowing for clearly defined, modular sensor network

components that can be virtualized, the two constituent elements hosted by the Operational Software unit form the basis of the proposed SDWSN framework.

By means of mimicking the flow of data occurring within the physical network, the ‘Virtual Network’ serves as a means to remotely monitor the logical facet of the dynamics within the same. This renders it as a platform to soft trial various network re-orchestration scenario(s) prior to actual implementation. By means of obviating the hardware requirement (for both debugging as well as testing of re-orchestration scenarios), cloud-hosted virtualization not only aids towards leaning out the network configuration process but also significantly expediting the same. It could also consist of certain functionalities such as display of serial output of the messages emanating from the virtual nodes along with their respective timestamps, allowing for gaining an estimate of the time period associated with various network operations and/or delay associated with network re-orchestrations, if any, beforehand.

The ‘Firmware Development and Configuration Unit’ consists of the necessary toolchains, source codes and compilers and offers the environment for development of the desired firmware files. It also consists of the necessary file conversion and configuration software that allows for uploading the code to the physical embedded node devices (and virtual nodes for IDEs that offer support for virtualization, as alluded to earlier).

The idea behind having in place a ‘Re-orchestration Management Unit’ is to tackle network fragmentation in a pre-emptive manner (, if possible) and resolve it (post-fragmentation) in accordance with a strategy involving the three different phases of re-orchestration, viz., ‘Data Analysis and Event-Identification’, ‘Re-orchestration-Planning’ and ‘Re-orchestration-Execution’. For this, it (i.e. The Re-orchestration Management Unit) houses the three sub-units, namely each of which house the respective dedicated knowledge components required for the corresponding phases of re-orchestration, elaborated within the following sub-section (section 4.7.1). The ‘Data Analysis and Event Identification’ sub-unit is responsible for detecting deviations from the normal service dataflow pattern (caused by occurrence of any network fragmentation event) and trigger the Re-orchestration-Planning’ phase into action. The ‘Re-orchestration-Planning’ sub-unit, on the other hand, is responsible for both initiation of measures towards gathering of information from the physical network layer required for planning the necessary re-orchestrations as well as for the actual ‘planning’ of the re-orchestrations to be assumed by the physical layer. Finally, the ‘Re-orchestration Execution’ unit is responsible for implementation of the outcomes of the planning process onto the actual physical network (via the aforementioned ‘Firmware Development and Configuration Unit’). These phases are elaborated within the following section (section 4.7).

4.7 Strategy for Network Re-orchestration

Cloud-governed Software-defined re-orchestration play(s) a key role in resolving sensor network fragmentation arising out events such as node-death, unforeseen node device malfunction, departure of a mobile router node beyond the communication range of its ‘children’ nodes, resulting in loss of connectivity to some parts of the network. It is deemed viable to ‘split’ the overall ‘re-orchestration’ process into three different phases viz.,

1. Data analysis and Event-Identification phase: analysis of real time network status data and tracking of important events. ,
2. Re-Orchestration Planning phase: upon identification of a significant event, follow working a plan for reacting to the event through possible software re-orchestration and lastly,
3. Re-Orchestration Execution phase: execution of re-orchestration plan.

These are briefly elaborated upon in the following paragraph but are described in detail with the help of an example using Cooja network simulator in section 5.7.2 within chapter 5.

4.7.1 Three-Phase WSN Re-orchestration Strategy

Real-time monitoring of the underlying physical WSN via the cloud-hosted virtual network may help reveal any important event that could potentially result in partial or complete disruption of the WSN operation. Events such as drifting of a mobile router node away from the WSN connectivity chain or any (static) router node on the verge of dying owing to depleted battery levels, necessitate requisite network re-orchestration to maintain network connectivity and ensure the continuity of flow of data across the same. Such potential network fragmentation events are continuously monitored by a dedicated ‘knowledge component’ (hosted by the ‘Data and Knowledge’ repository) during the ‘Data Analysis and Event-Identification’ phase. Once any event requiring network re-orchestration is detected by the said knowledge component, an alert or alarm is triggered so as to initiate the ‘Re-orchestration-Planning’ phase.

The Re-orchestration-Planning’ phase entails collection of the required information from the underlying physical WSN in a proactive manner. These pieces of information are utilized within the election process to determine the most suitable replacement router. Via delivering the outcome of the election process (i.e., identifying the suitable router node to take up the role of a replacement router), the ‘Re-orchestration-Planning’ phase sets the stage for the re-orchestration of the physical WSN. Progression of the overall re-orchestration process in accordance with this particular strategy ensures that the downtime or re-orchestration latency experienced by the network, if any, is confined solely to the ‘Re-orchestration-Execution’ phase.

The process of software-defined sensor network re-orchestration may entail a certain unavoidable amount of latency. This latency, which could be referred to as ‘network downtime’ pertains solely to the time consumed during the ‘Re-orchestration Execution’ phase. It is, however, worthwhile to analyse and formulate a generic model for the overall or end-to-end re-orchestration process, taking into account the latencies incurred in all of the three phases of re-orchestration viz., Data Analysis and Event-Identification phase, Re-orchestration-Planning phase and the Re-orchestration-Execution phase.

While latency incurred on account of the Re-orchestration Planning phase tends to relate to the number of messages that need to be exchanged amongst the relevant nodes (in accordance with the re-orchestration strategy derived and set apart during the same), the latency associated with the ‘Re-orchestration Execution’ phase depends on a host of factors including the topological organization of the sensor network and related paths for communicating and confirming the necessary changes.

4.7.2 Formulation of Generic Model for WSN Re-orchestration Latency

The key constituent parameters, along with their respective notations (wherever necessary), that in some way or the other, could influence the latency associated with the re-orchestration process, are aggregated, and enlisted as below:

Notation	Parameter represented
s	number of radio messages transmitted (by the node) per second
m	No. of radio message transmissions exchanged amongst various nodes (involved in the re-orchestration process) throughout the re-orchestration execution process (i.e., until the resumption of normal dataflow)
P_{size}	Size of the packet being transmitted by a node
n	number of nodes involved
h	average number of hops or hop count
t_{interval}	Pre-defined fixed time interval between two consecutive time slots (when 'scheduled-based channel access method is adopted)
t_{Tr}	Transmission time
t_{prop}	Propagation time
t_{sw}	Time required for network node to switch over from one functional role to another
L_{hop}	Single hop latency
$L_{\text{S-R}}$	Latency entailed by a single radio message to traverse from the sender or 'transmitter' node to the 'receiver node' (during multi-hop radio message transmission)

Herein, it is assumed that the network operates under relatively low-load conditions (i.e., the volume of data flowing across the system remains low at all times). Since the system as a whole does not get overloaded to an extent that exceeds beyond its capacity (to cope with the same) under any circumstances (even when operating in CSMA mode or under the influence of any random-access protocol), the actual channel throughput has not been taken into account. Also, it is assumed that the ' L_{hop} ' is always than ' t_{interval} '.

Consider a tree-topological sensor network organization, composed of the three core functionalities (i.e., Gateway, Router, and leaf node), as shown in figure 4-9. Let us suppose that owing to a special condition, a certain mobile router ' R_1 ' present within the network begins to irreversibly move away from its neighbouring routers and its constituent leaf member nodes. Unless the network were to undergo re-orchestration to replace the departing router with a new one (prior to it venturing out of communication range of both the leaf nodes within its cluster as well as the immediate lower-level router i.e., ' R_2 '), the rest of the dependent network clusters would be rendered disconnected from the gateway node i.e., ' G ' and thereby the cloud server.

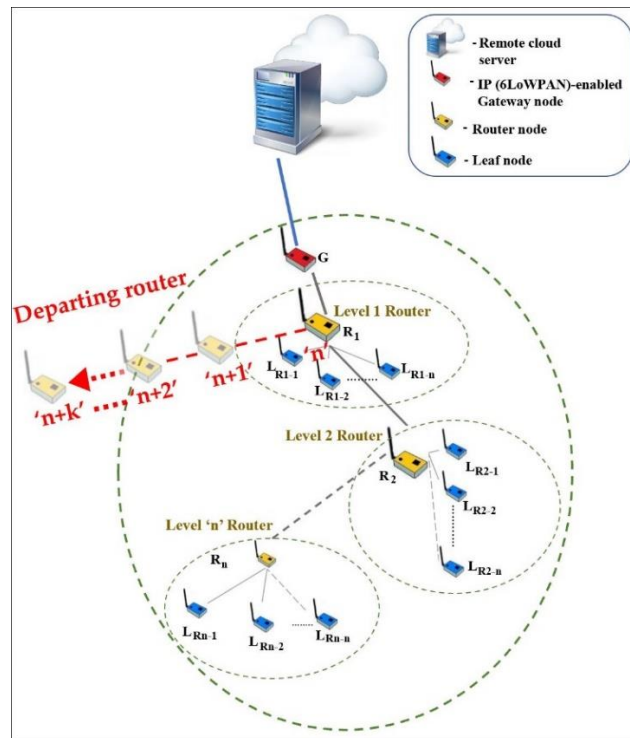


Figure 4-9 Generic schematic of a Typical WSN.

In the face of such imminent router departure, the network ensues upon undergoing the aforesaid three re-orchestration phases to pre-empt and avert such a massively disruptive network fragmentation event. Also, let us suppose that the outcome of the election process requires leaf node ‘ L_{R1-1} ’ to undergo functional re-orchestration (to say, take up the role of a replacement router).

As a means to gauge the impact of the above re-orchestration on the network in terms of the latency entailed, the time elapsed from the instant at which the leaf node ‘ L_{R1-1} ’ undergoes functional change (to say, take up the role of a replacement router) to the instant of time at which data from node ‘ L_{Rn-n} ’ reaches the gateway node ‘ G ’ (marking resumption of the normal service dataflow within the network), is measured.

First and foremost, latency entailed during a single hop must be delved into. Typically, it could be expressed as the summation of transmission, processing, queuing, access, propagation, forwarding and reception times.

Herein, for the sake of simplicity, only transmission and propagation times are taken into account. Thus, the mathematical equation to determine the delay incurred during a single hop ‘ t_{hop} ’ could be expressed as below.

$$L_{hop} = t_{tr} + t_{prop},$$

where, $t_{tr} = (P_{size}/s)$,

A radio message transmission emanating from a sender or transmitter node may have to traverse through a number of nodes prior to reaching a the intended ‘receiver’ node, (thus) entailing multiple hops. In case of such multi-hop radio message transmission, latency entailed by a single radio message to traverse from the sender or transmitter node to the ‘receiver node’,

$$L_{S-R} = h \times L_{hop}$$

where, h = No. of hops and

$$L_{hop} = \text{Single hop latency}$$

Number of hops i.e., ‘ h ’, in turn, depends on the number of nodes (present between the sender or transmitter node and the ‘receiver node’) to be traversed through by the radio transmission message. Mathematically,

$$h = n-1$$

$$\text{Therefore, } L_{S-R} = (n-1) \times L_{hop}$$

Herein, the progression of the re-orchestration process is based on the premise or assumption that all the messages get exchanged in a sequential fashion (only one message at a time), regardless of the channel access method adopted. Now, the time interval between two sequential message transmissions depends on the channel access method in place. While it is equivalent to the duration of time interval defined between the time slots allotted for two consecutive nodes for schedule-based channel access methods (such as TDMA or polling), it corresponds to the message (or Data) communication rate for ‘contention-based’ channel access methods (such as CSMA).

Thus, the cumulative latency entailed by the total number of sequential radio message transmissions exchanged amongst various nodes involved in the re-orchestration process, $L_{Cumulative}$

$$= (L_{S-R} + t_{interval}) \times m \dots \dots \dots (\text{for ‘schedule-based’ ‘channel access methods’ such as TDMA})$$

$$= (L_{S-R} \times s \times m) \dots \dots \dots (\text{for ‘contention-based’ ‘channel access methods’ such as CSMA})$$

Finally, the total re-orchestration latency i.e., the latency entailed by the re-orchestration execution process is the sum of the cumulative latency and the time required for node(s) (‘ L_{R1-1} ’ in this case as assumed earlier) to undergo functional re-orchestration.

Mathematically, it can be expressed as below.

$$L_{Re-orch_Exec} = L_{Cumulative} + t_{sw}$$

$L_{Re-orch_Exec}$ when schedule-based channel access method (such as Polling or TDMA) is adopted	$L_{Re-orch_Exec}$ when contention-based channel access method (such as CSMA) is adopted
$= [(L_{S-R} + t_{interval}) \times m] + t_{sw}$	$= (L_{S-R} \times s \times m) + t_{sw}$
$= \{[(h \times L_{hop}) + t_{interval}] \times m\} + t_{sw}$	$= \{[(h \times L_{hop}) \times s \times m] + t_{sw}$
$= \{ \{[(n-1) \times L_{hop}] + t_{interval}\} \times m\} + t_{sw}$	$= \{[(n-1) \times L_{hop}] \times s \times m\} + t_{sw}$
$= \{ \{[(n-1) \times (t_{tr} + t_{prop})] + t_{interval}\} \times m\} + t_{sw}$	$= \{[(n-1) \times (t_{tr} + t_{prop})] \times s \times m\} + t_{sw}$
$= \{ \{ \{[(n-1) \times [t_{tr} + (P_{size}/s)]] + t_{interval}\} \times m\} + t_{sw}$	$= \{(n-1) \times [t_{tr} + (P_{size}/s)] \times s \times m\} + t_{sw}$

Albeit at a high level, this model could offer an estimate of the downtime that may be experienced by a sensor network during the re-orchestration. Certain limitations associated with this model include assumption that each message tick represents the same amount of time duration. The exact time associated with any computation ensuing within the cloud during the Re-orchestration Planning phase may only be through simulation environments that have provision for message timestamps (so that the actual time engulfed for computation purposes could be worked out) within a simulation or actual physical hardware.

4.8 Conclusion

It is anticipated that the proposed approach will significantly aid the accrual of network flexibility via optimizing the process of network re-orchestration by curtailing the time associated with testing (via exploiting all the possible accessible node operational parameters to derive the necessary re-orchestrations). One of the key aspects associated with the proposed organization pertains to the collaborative engagement between a software library of reusable modules i.e., ‘Data and Knowledge Repository’ and virtualization environment at the cloud layer (of the proposed architecture) could facilitate for the necessary software-controlled switching, reassembly and disassembly of the requisite software modules onto the virtual nodes. Furthermore, the operational metrics pertaining to any of the software modules could also be subjected to manipulation prior to offloading onto a virtual node. This, in effect, opens the door for interacting with various scenarios related to off the shelf components or complete solutions to evolvable components and solutions that dynamically resides within the existing repository at the cloud.

This, in turn, can be accomplished via exercising dynamic software control over the several operational parameters available for software manipulation at both the node and network levels. Cloud-level virtualization and its flexible manipulation through software control offers an avenue for such flexible reformulation of core sensor network functionalities.

Software-controlled virtualization, working in conjunction with aforementioned ‘software-library’, offers an avenue for soft-trialling of network performances prior to implementation onto the Physical layer. By means of observing the implications of numerous software-defined alterations within the virtual environment, the most suitable functional configuration to be implemented on the node can be adaptively converged upon whilst exploring the degree of freedom procurable from each of the functionalities.

In essence, the proposed research work envisages an organization that continually monitors the operational dynamics of the underlying sensor network and identifies any (potential) situation necessitating re-orchestration beforehand (during the ‘Data Analysis and Event Identification’ phase). Upon foreseeing of any such event or scenario, it engages the virtualization unit to proactively interact with the Physical layer so as to gather the information required for the subsequent phase of ‘Re-orchestration Planning’. During the ‘Re-orchestration-Planning’ phase, the proposed organization ensues works out the most suitable ‘re-orchestrations’ to be applied onto the onto the Physical layer.

Such an organizational workflow aimed at pre-emptive planning of the re-orchestrations to be applied onto the physical layer would tend to confine the downtime suffered by the network to only the ‘Re-orchestration-Execution’ phase.

Chapter 5

SDWSN Concept Simulation and Testing

5.1 Introduction

This chapter details information (on the efforts undertaken) towards implementation and testing of the proposed SDWSN concept to render sensor networks capable of undergoing flexible network re-orchestrations. By means of delving into the specificities of implementations entailing certain example cases of software-defined re-orchestrations and offering incremental results in some of those cases, this chapter also reflects the viability of Contiki as a tool for the necessary software development and dynamic re-orchestration for both the physical as well as virtual environments within the proposed architecture.

Performance-related aspects arising out of the re-orchestration process viz., improvement with respect to a certain performance (such as packet loss mitigation), downtime suffered by the network have been highlighted within this chapter. The role of Contiki IDE to allow for programming of the Contiki-ported Texas Instruments CC2538 hardware nodes as well as the Cooja virtual nodes in a modular way, enabling dynamic flexible re-orchestration (of their functional role) via software control has been discussed.

5.2 Contiki Tool for WSN Softwarization Pre-requisites

By means of allowing for both firmware development as well as virtualization, the Contiki software, as a tool, tends to seemingly meet the key pre-requisites outlined in chapter 4. The firmware for each of the well-defined, core functional components (viz., Leaf, Router and Gateway functions, encompassing their respective operational components) so identified can be developed within the IDE (Integrated Development Environment) offered by the Instant Contiki – 2.7 software and rendered as a re-usable module. These re-usable firmware modules so developed could then be compiled and used for configuring both physical and virtual nodes via Contiki-based software. By means of facilitating for a virtualization environment, Contiki's Cooja network 'simulation tool' fulfils the crucial requisite of running soft trials of network re-orchestration scenarios to converge upon the most suitable re-orchestrations to be applied onto the physical WSN.

5.3 Contiki-Based Pseudo Codes for Key Modular WSN Components

5.3.1 Contiki-Based Pseudo Code for WSN Leaf Function

The pseudo code for the firmware for the WSN Leaf function is as depicted in figure 5-1 below. It consists of the various sub-modules, consisting of their respective constituent operational parameters. These operational parameters could be subjected to software manipulation in a bid to realize the desired node-operational re-orchestrations (pertaining to the leaf function). The design of the pseudo code model adopted herein encompasses the various operational activities catered to by the leaf function in a chronological order (ranging from selection of the desired sensor to the transmission of the data sensed by it in accordance with the channel access method adopted).

Manipulation of the node-operational components via the Contiki-based firmware used for configuring the TI CC2538 node to execute 'leaf' functional role

- 1: Sensor selection:** Execute the light, temperature and RSSI sensor function(s) individually (one at a time) or together (multiple at a time) and acquire the desired sensor readings

```
light_dbl=adc_sensor.value(ADC_SENSOR_ALS); //Light sensing function
```

```
temp = adc_sensor.value(ADC_SENSOR_TEMP); //Temperature sensing function
```

```
rssl=packetbuf_attr(PACKETBUF_ATTR_RSSI); //RSSI sensing function
```

- 2: Buffering of sensed data:** Store the data sensed viz., light temperature and RSSI variables within an array variable (of four elements) say, 'c[4]')

```
c[0]=node_ID; // Assigning the ID of the node to the first element of the array  
declared
```

```
c[1]=light; //Assigning the light value sensed to the second element of the array  
declared
```

```
c[2]=temp; //Assigning the temperature value sensed to the third element of the  
array declared
```

```
c[3]=rssl; //Assigning the RSSI value sensed to the fourth element of the array  
declared
```


7: Enabling a channel access method: Whilst sensing data, the leaf node continuously checks if the counter value broadcasted by the governing sink node is the same as its node ID. If it is the same, a variable, say, Transmit_Flag (initially assigned with the value '0') is 'set' (i.e., assigned value '1')

```
if(received_node_ID[0] == node_ID_of_receiver_node) // Check if incoming
counter value is the same as node ID
```

```
    {
        Transmit_Flag=1;    // Set 'Transmit flag'
    }
else
    {
        Transmit_Flag=0;
    }
```

For 'Polling' mode: The status of the Transmit_Flag' variable is checked. If it is 'set', the array to be sent is transmitted.

```
PROCESS_THREAD(cc2538_demo_process, ev, data)
{
    while(1) {
        if(Transmit_Flag==1) //For 'Polling' mode
        {
            packetbuf_copyfrom(&c, sizeof(c); //where c is array of sensed
                                                    data variables to be
                                                    transmitted
            broadcast_send(&bc);

            Transmit_Flag=0; //Resetting the 'Transmit flag'
        }
    }
}
```

For 'CSMA' mode: The status of the 'Transmit_Flag' variable is not checked. The array to be sent is transmitted regardless of the existing status of the 'Transmit_Flag' variable.

```
while(1) { // If 'Transmit flag' =0 or if a requisite external command
            message directing it to execute in CSMA mode is received,
    {
        packetbuf_copyfrom(&c, sizeof(c);
        broadcast_send(&bc);
    }
}
```

Figure 5-1 Pseudo code for the (Contiki-based) firmware used for configuring TI CC2538 with the WSN Leaf function.

5.3.2 Contiki-Based Pseudo Code for WSN ‘Routing’ Function

The pseudo code for the firmware for the WSN Routing function is as depicted in figure 5-2. It consists of the various sub-modules, operational parameters within which could be tapped into towards realizing the desired node-operational re-orchestrations (pertaining to the ‘routing’ function). The design of the pseudo code model adopted herein encompasses the various operational activities catered to by the routing function in a chronological order (ranging from initialization and reception of the incoming sensed data up to relaying it another router or a Gateway node).

Manipulation of the constituent operational components via the Contiki-based firmware used for configuring the TI CC2538 node to execute ‘router’ functional role

1: Initialization of arrays: Declaring and initializing of arrays of a certain size ‘r’ to receive sensed data variables, say, light, temperature and RSSI,

```
var_1_light[r]      = {01,02,....0r};
```

```
var_2_temperature[r] = {01,02,....0r};
```

```
var_3_rssi[r]       = {01,02,....0r};
```

from all the leaf nodes within its cluster)

2: Receive sensed data: Receiving and storing the incoming sensed values emanating from the leaf nodes on a node-by-node basis.

```
int16_t *temp_data_pointer;
```

```
temp_data_pointer = (int16_t *)packetbuf_dataptr();
```

```
x          = temp_data_pointer[0]; //Receive node ID , say ‘x’ of the leaf node
```

```
light[x]   = temp_data_pointer[1]; //Receive light value transmitted by node ‘x’
```

```
temperature[x] = temp_data_pointer[2]; //Receive temperature value variable  
transmitted by node ‘x’
```

```
rssi[x]    = temp_data_pointer[3]; //Receive RSSI value transmitted by node ‘x’
```

3: Enabling a channel access method: The firmware used for configuring a CC2538 SoC to act as a ‘polling’ router consists of a ‘counter’ variable (initially set to ‘0’) that gets incremented (in accordance with the value specified within its ‘etimer’ function). After each such increment, this value is broadcasted by the router node to all the leaf nodes within its cluster. The counter variable ceases to increment once its value is equal to the number of leaf nodes within its cluster (pre-defined within its firmware).

```
while(1) {

    PROCESS_YIELD();
    if(ev == PROCESS_EVENT_TIMER)
        {

            ‘counter’++;
            a_[0]=counter;
            a_[1]=TIME_IN_SEC - 1;
            a_[2]=1;

            etimer_set(&et, CLOCK_SECOND*‘N’);

            /*Packets are transmitted (broadcasted) using this function*/

            packetbuf_copyfrom(&a, sizeof(a)); // where ‘a’ is an array variable
                                                containing the counter values to
                                                be broadcasted as one of its
                                                elements.

            broadcast_send(&bc);

                if(counter==Number of leaf nodes within its cluster)
                {
                    counter=(counter % Number of leaf nodes within its cluster);
                }
        }
}
```

4: Channel Allocation: The TI CC2538 nodes could be configured to communicate via one of the available channels from 11 to 26 via entering the desired numerical value within the following line of code

```
#define CC2538_RF_CONF_CHANNEL <Required value from 11 to 26>
```

5: Buffering of data variables to be transmitted: Buffering the sensed data received within an array variable say, 'e' (of four elements)

```
e[0]= node_ID_number;
e[1]= light_v;
e[2]= temperature_v;
e[3]= rssi_v;
```

6: Radio Transmission Power: The desired radio transmission power mode (of the available power modes) can be set via entering the hexadecimal value corresponding to that power mode (refer Table 5-1) within the following 'cc2538_rf_power_set' function

```
cc2538_rf_power_set(uint8_t new_power)
{
    REG(RFCORE_XREG_TXPOWER) = new_power;
    return (REG(RFCORE_XREG_TXPOWER) & <hexadecimal_value>);
}
```

7: Setting data communication rate: The desired data communication rate can be set via specifying the particular value (say,) 'x' within the 'etimer' (i.e., 'event timer') function'

```
etimer_set(&et, CLOCK_SECOND*(1/x));
```

8: Relay (requisite) stored data: In order to transmit the array of data over to the CC2538 SoC configured as a gateway node, the name of the array variable has to be specified within the following function(s):

```
packetbuf_copyfrom(&e, sizeof(e); // 'e' is the array variable to be transmitted to the
sink node
broadcast_send(&bc);
```

Figure 5-2 Pseudo code for the (Contiki-based) firmware used for configuring TI CC2538 with the WSN Router function.

5.3.3 Contiki-Based Pseudo Code for WSN Gateway Function

Implementation of the gateway unit involved utilizing a TI CC2538 Evaluation module (EM) in conjunction with the Raspberry Pi 3 (Model B V1.2) device, as depicted in figure 5-3. Herein, the CC2538 EM was configured with the gateway firmware allowing it to accommodate for the data relayed by the router-configured nodes. The Raspberry Pi 3, on the other hand, has a ‘python’ script running within it, enabling it to post the data (so acquired by the ‘Gateway-firmware’ configured TI CC2538 node) over to the local server. Internet connectivity is provisioned to the Raspberry Pi unit via ethernet. The python script firstly reads the data acquired by CC2538 EM connected to it via USB over serial communication. It then checks for the integrity of the data via reading for instances of the data read and posts it to the local server using the REST API. Besides the advantages pertaining to cost and form factor i.e., compactness, such a Raspberry Pi-provisioned gateway setup allows for a ‘readily portable’ solution, to some extent, given the multitude of interfaces for availing Internet connectivity (ethernet, onboard-Wi-Fi, Bluetooth, etc.)

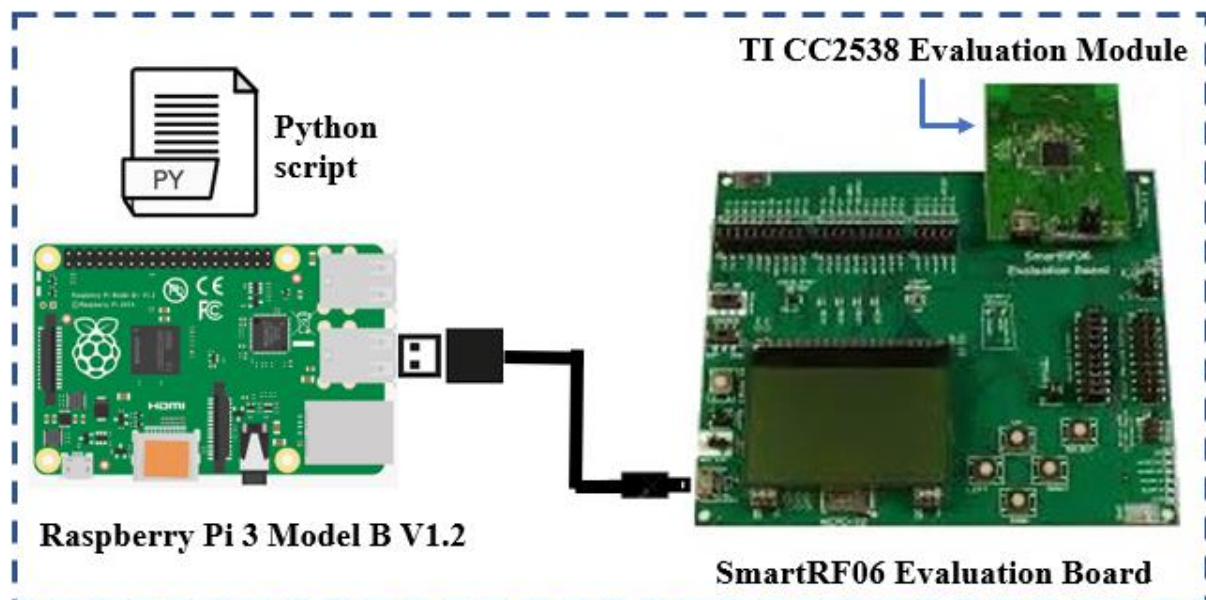


Figure 5-3 Gateway unit realized using TI CC2538 Evaluation module (EM) in conjunction with the Raspberry Pi 3 (Model B V1.2) device.

The pseudo code for the firmware for the WSN gateway function is as depicted in figure 5-4. It consists of the various sub-modules, operational parameters within which could be tapped into towards realizing the desired node-operational re-orchestrations (pertaining to the Gateway function). The design of the pseudo code model adopted herein encompasses the various operational activities catered to by the Gateway function in a chronological order (ranging from initialization and reception of the incoming sensed data up to subjecting it through the requisite protocol conversion so as to relaying it over to the remote cloud-server over the Internet).

<p>Implementation of the operational components within the Contiki-based firmware and the python script developed for TI CC2538 and Raspberry Pi 3 Model B V1.2 respectively,</p>
<p><u>Texas Instruments CC2538 Evaluation Module (EM)</u></p>
<p>1: Initialization of arrays: Declaring and initializing of arrays of a certain size ‘m’ to receive all sensed data variables, say light, temperature and RSSI, relayed by an intermediate CC2538 router node</p> <pre> var_1_light[m] = {0₁,0₂,...,0_m}; var_2_temperature[m]= {0₁,0₂,...,0_m}; var_3_rssi[m] = {0₁,0₂,...,0_m}; </pre> <p>(depending upon the number of leaf nodes within its cluster)</p>
<p>2: Receive sensed data: Receiving data from CC2538 router nodes and storing them in a specific format (node-by-node basis for all the leaf nodes)</p> <pre> int16_t *temp_data_pointer; temp_data_pointer= (int16_t *)packetbuf_dataptr(); x = temp_data_pointer[0]; //Receive node ID , say ‘x’ of the leaf node light[x] = temp_data_pointer[1]; //Receive light value transmitted by node ‘x’ temperature[x] = temp_data_pointer[2]; //Receive temperature value variable transmitted by node ‘x’ rssi[x] = temp_data_pointer[3]; //Receive RSSI value transmitted by node ‘x’ </pre>
<p>3: Channel Allocation: The TI CC2538 nodes could be configured to communicate via one of the available channels from 11 to 26 via entering the desired numerical value within the following line of code:</p> <pre> #define CC2538_RF_CONF_CHANNEL <Required value from 11 to 26> </pre>
<p>4: Read data over serial communication: Receive data from node acting as gateway over serial communication via python script</p>
<p>5: Check data integrity: Check integrity of received data via comparing lengths of certain number of ‘strings’ of the data</p>
<p>6: Push data to remote server: Push data over to a remote-server using the ‘GET command’ along with requisite ‘URL’ of (server) webpage.</p>

Figure 5-4 Pseudo code for the (Contiki-based) firmware used for configuring TI CC2538 with the WSN Gateway function.

The python script within Raspberry Pi using the 'REST API' to read serial data from TI CC2538 and escalate data to the remote cloud server is as depicted in figure 5-5.

Python script within Raspberry Pi using the REST API to read serial data from TI CC2538 and escalate data to the remote cloud server

```
import serial
import os
import urllib
import urllib2
import webbrowser

while True:
    ser = serial.Serial('/dev/ttyUSB1',baudrate=115200)
    print ser
    v1= ser.readline()
    v2= ser.readline()
    v3= ser.readline()
    v4= ser.readline()

    i1=len(v1)
    i2=len(v2)
    i3=len(v3)
    i4=len(v4)

    ser.close()

    if i1==i2:
        x=v1
    elif i2==i3:
        x=v2
    elif i3==i4:
        x=v3
    elif i1==i3:
        x=v3
    elif i1==i4:
        x=v4
    elif i2==i4:
        x=v2
```

```

elif i1<220 and i2<220 and i3<220 and i4<220:
    continue
else:
    continue
print x

b = x
print b
url = 'http://sense.aut.ac.nz/MAC/TI/Ethernet/TI_12Aug18_T1.php'
b = '|' + b

values = {'TheString' : b}
data = urllib.urlencode(values)
req = urllib2.Request(url,data)
response = urllib2.urlopen(req)

the_page = response.read()
print "Data Sent to Server Successfully!!!"

```

Figure 5-5 Python script within Raspberry Pi using the ‘REST API’ to read serial data from TI CC2538 and escalate data to the remote ‘cloud server’.

5.4 Node and Network-Level WSN Re-orchestrations

Specificities pertaining to the exertion of software control over certain constituent node-operational parameters present within the typical, say, leaf-node firmware such as ‘Sensor Selection’, ‘Data Acquisition’, ‘Buffering of Sensed Data’, ‘Data Computation’, ‘Channel Allocation’, ‘Radio Transmission Power’ and ‘Data Communication Rate’ via Contiki IDE (Integrated Development Environment) are elaborated below. Prior to delving into the discussions pertaining to each such node-operational parameter, an integrated pseudo code encompassing for the same has been depicted in figure 5-6. The complete code is available within section A.1 of the appendix. As a means to reflect the process of ‘software-defined’ ‘re-orchestration’ through examples involving the aforementioned parameters, two incremental examples cases (pertaining to node and network levels) are discussed.

5.4.1 Integrated (Contiki-Based) Pseudo Code

Example unified firmware used for configuring a node that is capable of accommodating for and executing operational activities pertaining to both router and ‘leaf functional roles

Initialization of arrays:

Declare and initialize one or multiple arrays (if need be, depending upon the number of types of incoming sensed variables) of a certain size (depending upon the number of leaf nodes within its cluster) to receive all the incoming sensed data variables

Declaration and initialization of arrays of a certain size 'r' to receive sensed data variables, say, light, temperature and RSSI,

```
var_1_light[r]      = {0,0,....0r};
```

```
var_2_temperature[r] = {0,0,....0r};
```

```
var_3_rssi[r]      = {0,0,....0r};
```

from all the leaf nodes within its cluster)

If external radio message flag received directing execution of the 'leaf function' sub-modules

Sensor selection:

Execute the desired sensor function(s) (one or more) and acquire the sensor readings

```
light_dbl=adc_sensor.value(ADC_SENSOR_ALS); //Light sensing function
```

```
temp = adc_sensor.value(ADC_SENSOR_TEMP); //Temperature sensing function
```

```
rssi=packetbuf_attr(PACKETBUF_ATTR_RSSI); //RSSI sensing function
```

If external radio message flag received directing execution of the router function sub-modules

Receive sensed data: Receive and store incoming sensed data from all constituent leaf nodes on a node-by-node basis

```
int16_t *temp_data_pointer;

temp_data_pointer= (int16_t *)packetbuf_dataptr();

x          = temp_data_pointer[0]; //Receive node ID , say 'x' of the
          leaf node

light[x]   = temp_data_pointer[1]; //Receive light value transmitted by
          node 'x'

temperature[x] = temp_data_pointer[2]; //Receive temperature value variable
          transmitted by node 'x'

rssi[x]    = temp_data_pointer[3]; //Receive RSSI value transmitted by
          node 'x'
```

Buffering of data variables to be transmitted: Store received data within an array of requisite size

Buffering the sensed data received within an array variable say, 'e' (of four elements)

```
e[0]= node_ID_number;
e[1]= light_v;
e[2]= temperature_v;
e[3]= rssi_v;
```

If external radio message flag received directing execution of the 'leaf function' sub-modules

Channel Allocation: Set the node to communicate using (one of the available and) desired channel(s)

```
#define CC2538_RF_CONF_CHANNEL <Required value from 11 to 26>
```

Buffering of sensed data: Store the sensed data within an array of requisite size along with node ID

Store the data sensed viz., light temperature and RSSI variables within an array variable (of four elements) say, 'c[4]')

```

c[0]=node_ID; // Assigning the ID of the node to the first element of the array
              declared

c[1]=light; //Assigning the light value sensed to the second element of the
            array declared

c[2]=temp; //Assigning the temperature value sensed to the third element of
           the array declared

c[3]=rssi; //Assigning the RSSI value sensed to the fourth element of the
           array declared

```

Radio Transmission Power: Set the node to transmit data using (one of the available and) desired channel(s)

The desired radio transmission power mode (of the available power modes) can be set via entering the hexadecimal value corresponding to that power mode (refer Table 5-1) within the following ‘cc2538_rf_power_set’ function

```

cc2538_rf_power_set(uint8_t new_power)
{
    REG(RFCORE_XREG_TXPOWER) = new_power;
    return (REG(RFCORE_XREG_TXPOWER) & <hexadecimal_value>);
}

```

Setting data communication rate: Set the desired rate of transmission of data (via specifying the particular numerical value within the requisite function

The desired data communication rate can be set via specifying the particular value (say,) ‘x’ within the ‘etimer’ (i.e., ‘event timer’) function’

```

etimer_set(&et, CLOCK_SECOND*(1/x));

```

Transmission of sensed data: Transmit the array of sensed data to a ‘sink’ node (via entering the name of the array variable to be transmitted within the requisite function)

```

packetbuf_copyfrom(&c, sizeof(c); //’c’ is the array variable to be transmitted
                  to the sink node

broadcast_send(&bc);

```

Enabling a ‘channel access’ method: Enable a particular ‘channel access’ method viz., CSMA, TDMA, etc.

Whilst sensing data, the leaf node continuously checks if the counter value broadcasted by the governing sink node is the same as its node ID. If same, a variable say, ‘Transmit_Flag’ (initially assigned with the value ‘0’) is ‘set’ (i.e., assigned value ‘1’)

```
if(received_node_ID[0] == node_ID_of_receiver_node) // Check if incoming
counter value is the same as node ID
    {
        Transmit_Flag=1;    // Set ‘Transmit flag’
    }
else
    {
        Transmit_Flag=0;
    }
```

For ‘Polling’ mode: The status of the ‘Transmit_Flag’ variable is checked. If it is ‘set’, the array to be sent is transmitted.

```
PROCESS_THREAD(cc2538_demo_process, ev, data)
    {
        while(1) {
            if(Transmit_Flag==1) //For ‘Polling’ mode
                {
                    packetbuf_copyfrom(&c, sizeof(c); //where ‘c’ is array of
                                                    sensed data variables to
                                                    be transmitted

                    broadcast_send(&bc);

                    Transmit_Flag=0; //Resetting the ‘Transmit flag’
                }
        }
    }
```

For CSMA mode: The status of the ‘Transmit_Flag’ variable is not checked. The array to be sent is transmitted regardless of the existing status of the ‘Transmit_Flag’ variable.

```
while(1) { // If ‘Transmit flag’ =0 or if a requisite external command
```

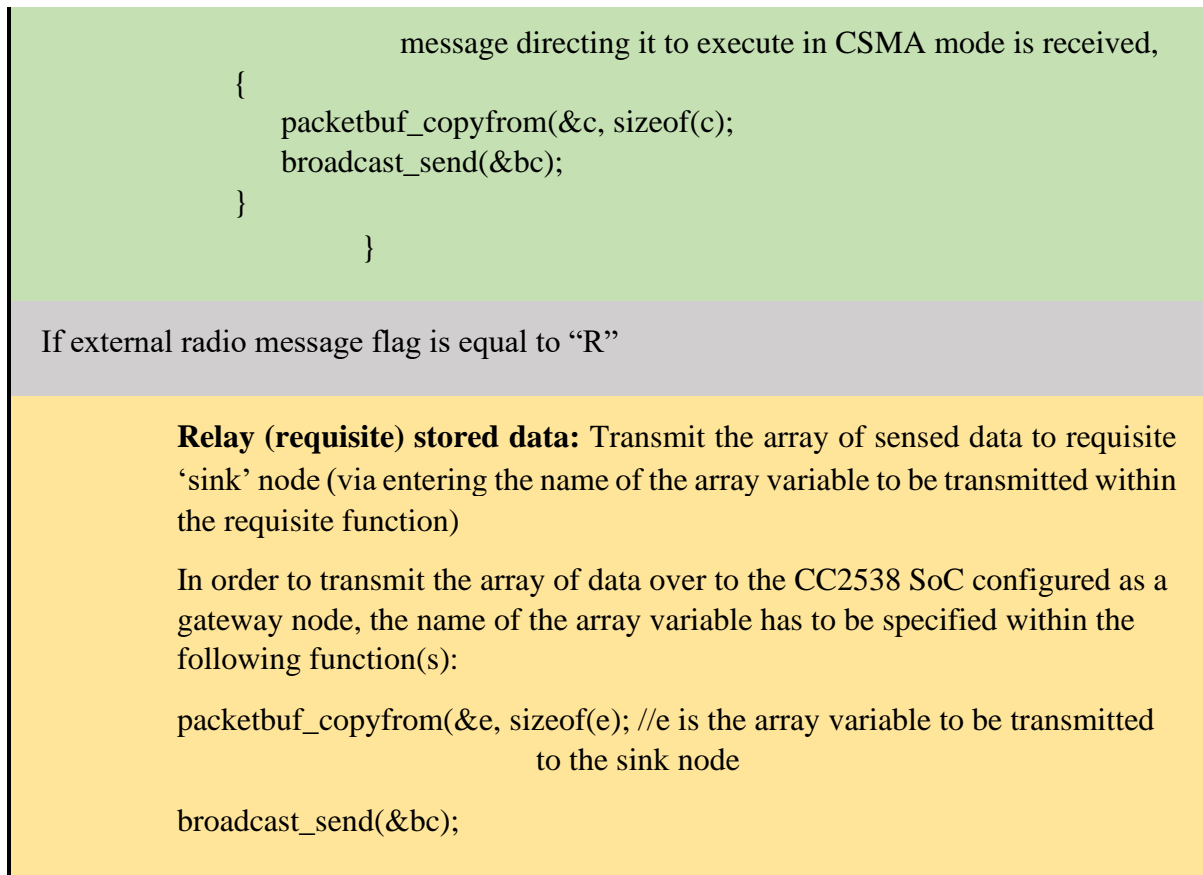


Figure 5-6 Example pseudo code for a (Contiki-based) unified firmware.

Certain such node-operational parameters present at the various layers of the communication stack viz., sampling rate, sensor selection, buffer size etc. present within the physical layer, communication protocols, channel allocation, etc., present within the MAC layer and so on, could be manipulated by means of tweaking the relevant functions of the firmware fed within the node as elaborated (for some of such operational parameters) below. As a means to reflect the saliency of Cooja in facilitating for soft trialling of various network re-orchestration scenarios, certain incremental example implementations (involving manipulation of certain parametric flexibilities discussed below) are provided.

5.4.2 Sensor Selection

TI CC2538 has a number of ADC (Analog-to-Digital) sensors associated with it such as those for light, temperature, RSSI, etc., offering for heterogeneity of sensing. Through ‘C’ codes written within Contiki IDE, each such sensor could be accessed via including the necessary header file (i.e., "dev/adc-sensor.h") and invoking the respective sensing functional module. For example, in order to measure ambient light, the functional module ‘adc_sensor.value(ADC_SENSOR_ALS)’ has to be incorporated or retained within the code. Similarly for selection of on-chip temperature and RSSI sensing variables, the functions ‘adc_sensor.value(ADC_SENSOR_TEMP)’ and ‘packetbuf_attr(PACKETBUF_ATTR_RSSI)’ ought to be invoked within the code. By means

of retaining the desired sensor functional modules and removing the unwanted sensor modules, software control could be leveraged upon to dynamically select and/or activate only the required sensing capabilities offered by the node. As a means to reflect the implementation pertaining to the above, relevant sections of code are provided as below.

```
Light          = adc_sensor.value(ADC_SENSOR_ALS);
temperature    = adc_sensor.value(ADC_SENSOR_TEMP);
rssi           = packetbuf_attr(PACKETBUF_ATTR_RSSI);
```

5.4.3 Buffer Size

A buffer could be created within the node via declaring an array of the desired (initial size) within the code fed into the node. Example declarations of such array variables of size 16 (i.e., capable of accommodating for 16 elements) within a code that has been used for configuring a sink node is provided below.

```
short signed light_[16]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed temperature_[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed rssi_[16]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

Herein too, the ‘#define’ directive could be employed to define its initial size as a ‘macro’ definition with a certain (initial) constant value associated with it. Since such macros so defined allow for the (pre-)assigned constant values to get declared throughout the ‘C’-based program present within the node, the size of the array could be dynamically manipulated via tweaking that particular value at run-time and subsequently re-compiling and re-configuring the code. For example, the size of an array named `uint16_t arlight[ARRAY_LENGTH_RT_LIGHT]`; declared within the code, could be dynamically manipulated via tweaking the value to the macro `ARRAY_LENGTH_RT_LIGHT` defined within the code.

In regard to the parametric flexibilities of sampling rate and buffer size available within the physical layer that could be wrought through Contiki-provisioned software control, an incremental example case of QoS-based dynamic node-level re-orchestration, depending upon the prevailing network conditions, is put forth.

5.4.4 Data Communication Rate

Via exerting software control over the ‘`etimer_set(&et, CLOCK_SECOND*(‘1/N’))`’ functional module,

where, ‘N’ is either the ‘communication rate’ or ‘reporting time’ interval or ‘servicing’ interval (and could either be a decimal or an integer value), the sampling rate and/or data reception rate of the node can be configured and/or dynamically manipulated as per the prevailing service requirement. For example, in order to configure a leaf node to transmit the data sensed by it to a nearby sink node at the rate of four packets/second, the ‘`etimer`’ function would need to be invoked as below.

```
etimer_set(&et, CLOCK_SECOND*(1/4)); i.e., etimer_set(&et, CLOCK_SECOND*0.25);
```

This could also be defined as a ‘macro’ within the code which allows for declaration of constant values (associated with it) throughout the program. Changing this value to another value results in changing the reporting time interval of the node to the new constant value wherever this ‘etimer’ function has been invoked within the figure. However, in both of the above ways, the code has to be re-compiled and re-fed into the node.

As a scenario, consider a virtual point-to-point network running within the Cooja virtualization unit wherein an end device transmits the sensed data (represented by the randomly generated values of the light, temperature and RSSI) to a coordinator (i.e., router or sink) node. In the event of occurrence of ‘significant’ dynamics within the monitored ‘phenomenon’ with respect to any of the parameters (say, occurrence of an event causing heavy fluctuations within the RSSI signals or significant deviation from the usual RSSI values), data-intensive sensing requirements necessitate higher sampling rates to capture more accurate information i.e., data pertaining to that particular unusual event. Figure 5-7 below shows the impact of exerting software control over node-operational parameters of the ‘communication rate’ or ‘reporting interval’ of any leaf node by means of running a test wherein the same virtual network is run for two different communication rates (for comparison purposes). Screenshot of the mote output window when the network is run at a low data communication rate is as depicted in figure 5-7 a. Herein, large time intervals between the ongoing communication amongst the two nodes (as can be observed from the timestamps associated with both the nodes at various instances of time) reflects the lower frequency of data communication between the two. However, upon re-orchestrating the individual node-operational parameter of ‘etimer’ within the firmware to run the network at a much higher communication rate (as indicated by the timestamps associated with both the nodes at various instances of time within figure 5-7b), more accurate information pertaining to the monitored event can be captured.

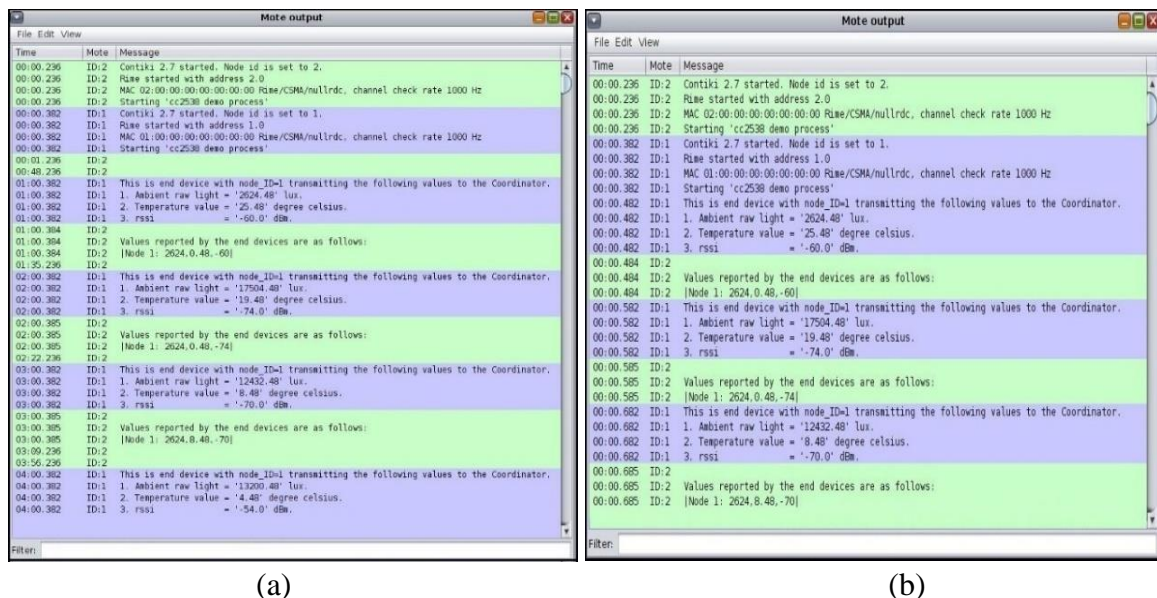


Figure 5-7 Dynamic re-orchestration of sampling rate through Contiki-provisioned software control to sample data at an increased rate for improved accuracy of (critical) data captured - (a) Lower data communication rate and (b) Higher data communication rate.

5.4.5 ‘Arrival’ and ‘Service’ Rates

Similar to that of the sampling rate, exertion of software control over the ‘`etimer_set(&et, CLOCK_SECOND*(‘1/N’))`’ functional module,

where, ‘N’ is either the ‘arrival and/or data transmission’ interval or ‘servicing’ interval (and could either be a decimal or an integer value), also enables dynamic configurability over data arrival (i.e., data transmission rate of the transmitter node) and data servicing (i.e., reception rate of the receiver node in consideration) rates. These too, could also be defined as a ‘macro’ within the code which allows for declaration of constant values (associated with it) throughout the respective programs to be fed the transmitter and receiver nodes. Changing this constant value to another constant value results in changing the data arrival of data servicing (reception) rates of the respective pertinent nodes (as per the new constant value entered) wherever this ‘`etimer`’ function has been invoked within the code. As mentioned earlier, both processes inevitably involve code re-compilation and feeding the executable format of the code so compiled into the respective nodes.

Equipping the IoT-WSN architectures to better regulate the flow of data in-sync with the dynamics of the physical phenomenon necessitates real-time manipulation i.e., ‘reconfiguration’ of the key ‘Physical Sensor Network’ (PSN) data acquisition ‘parameters’ such as buffer size and sampling rate. In this regard, an experiment involving a simplistic (two-node) point-to-point network was conducted within the virtual Cooja simulation platform, as shown in figure 5-8. Herein, one node was programmed to act as a transmitter whereas the other one was programmed to act as the receiver node (entrusted with the responsibility of servicing the data packets received from the transmitter node). Exertion of such software control of (either) the service rate (or the buffer size) of a receiver node (i.e., a node configured with a dominant routing or gateway function) is key to avoid packet loss of incoming data transmitted by a node configured with the ‘leaf’ (or router) function and maintenance of QoS whilst controlling the flow of sensed data. Herein, the ‘`CLOCK_SECOND`’ variable present within ‘`etimer_set`’ function of Contiki-based firmware ‘C’ code was varied (as mentioned in sub-section 5.4.4) via software control to gradually increase the service rates of the coordinator.

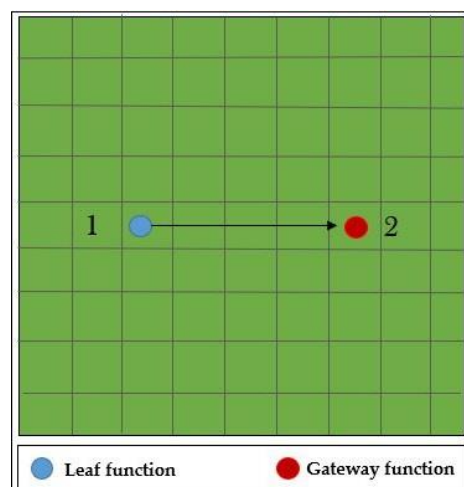


Figure 5-8 Virtual 2-node point-to-point network depicting the Cooja nodes 1 and 2 configured as the Leaf function and Gateway function nodes respectively.

The relevant section of the Contiki code associated with the transmitter node is provided as below. Herein, the sensed data-packets are transmitted to the receiver node along with the ‘node ID’ as well as the total number of packets sent.

```
while(1) {
    etimer_set(&et, CLOCK_SECOND*(1/N));
    sentcounter++;
    totalsentcounter=sentcounter;
    c[b-1]=totalsentcounter; // where ‘c’ is the array of sensed data packets and ‘b’ is
                             // the numerical value corresponding to the size of the
                             // buffer array
    printf("Total number of packets sent          = '%d' packets.\n",sentcounter);
    printf("Total number of packet elements sent = '%d' packets.\n",totalsentcounter);
    packetbuf_copyfrom(&c, sizeof(c));
    broadcast_send(&bc);
}
```

The relevant section of the Contiki code associated with the receiver node is provided as below. Herein, packet loss is determined via subtracting the received sensed data packets from the total number of packets sent (sent by the transmitter node).

```
static void
broadcast_rcv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    x =received_node_ID;
    data_variable_1[x] =dataptr_temp1[1];
    data_variable_2 [x] =dataptr_temp1[2];
    .....
    data_variable_n[x] =dataptr_temp1[n];
    packetsent[x]=dataptr_temp1[n+1];
    if (x==node_ID_of_receiver_node)
        { rcv_counter++;
```

```

packetrecv[x] = Packets received.
packetloss[x]=packetsent[x]-packetrecv[x];
}
}

```

Complete codes for both transmitter and receiver are placed within section A.2 of the appendix.

The main intention of this particular exercise was to study the effect of varying the service rate (SR) (i.e., the number of incoming data packets served per second) of the Cooja Gateway node 2 on the number of packets serviced, packets lost and its buffer usage, keeping the arrival rate (AR) i.e., the sampling rate of the Cooja leaf node 1, constant. The service parameter value used for configuring the gateway node was incrementally varied i.e., 1, 2, 4, 6, 8 and 10 packets/second whereas its buffer size was kept constant at 350 packets. The end device was configured to transmit its data to the gateway at a constant rate of 10 samples per second.

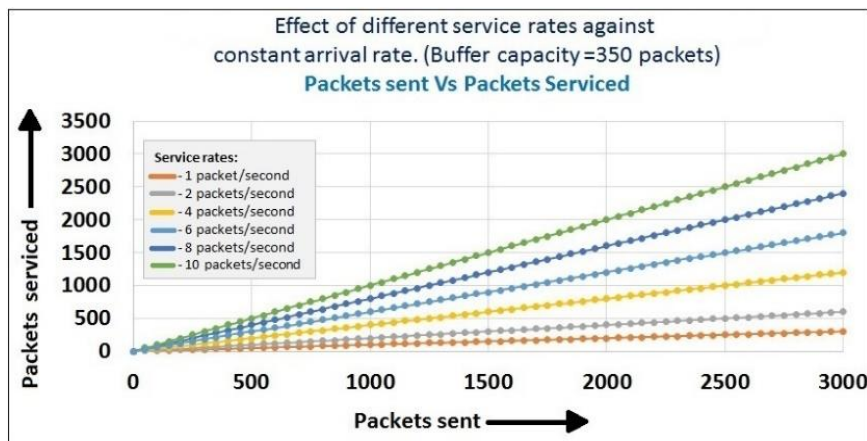


Figure 5-9. Impact of service rate-based node-level re-orchestrations on total incoming data packets serviced.

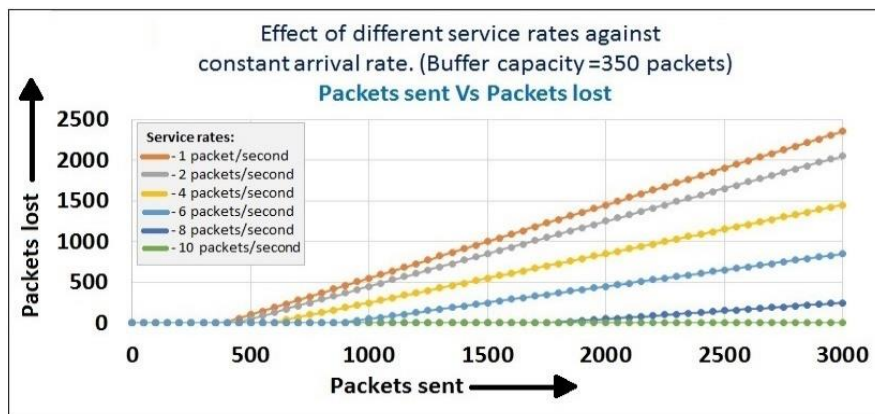


Figure 5-10. Impact of service rate-based node-level re-orchestrations on total incoming data packets lost.

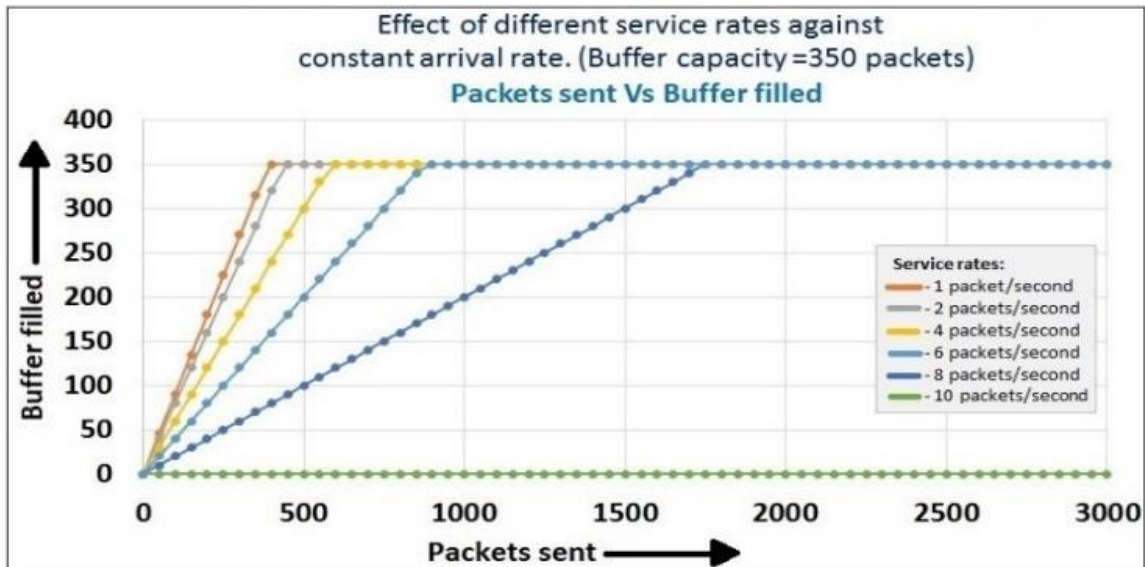


Figure 5-11. Impact of service rate-based node-level re-orchestrations on the gateway node buffer filled.

From figures 5-9 [42], 5-10 [42] and 5-11 [42], it can be clearly observed that when $SR=AR$, all the data packets are serviced, and none are lost. As the service rate is decreased progressively via implementation of the pertinent node-level re-orchestrations, lesser number of packets are serviced leading to faster buffer occupancy and a greater number of packets are lost.

In this particular experiment, the node-level re-orchestrations were first tested on the virtual environment offered by Cooja simulator. In this way, the most suitable node-level re-orchestrations can be determined so as to reconfigure the physical sensor nodes over the Internet to prevent packet losses and rapid buffer saturation.

5.4.6 Radio Transmission Power

The RF transmission power of the Contiki-ported CC2538 module can be manipulated via exerting software control over the 'TXPOWER' register within the `cc2538_rf_power_set(uint8_t new_power)` function (present within the 'cc2538-rf.c' file) by means of Contiki. The relevant section of the for the same is as provided below.

```
return (REG(RFCORE_XREG_TXPOWER) & 0x42);
```

Although a host of different register value settings can be specified to adjust the transmission output power (to avail transmission powers ranging from 7.5 dBm to 22 dBm) as depicted in Table 5-1 below, it is recommended that the CC2538 transceiver be configured to operate only on certain recommended ('TXPOWER' register) settings amongst those.

Table 5-1 Table showing the transmission powers that can be selected to configure the TI CC2538 wireless transceiver and their corresponding hexadecimal values to be used within relevant section of the code for the same.

RF Output Transmission Power (dBm)	Hexadecimal Value (TXPOWER)
22	0xFF
21.5	0xED
20.9	0xDF
20.1	0xC5
19.6	0xB6
19	0xB0
17.8	0xA1
16.4	0x91
14.9	0x88
13	0x72
11	0x62
9.5	0x58
7.5	0x42

5.4.7 Channel Access Method

Channel access methods available at the MAC layer i.e., TDMA or polling, CSMA, TDMA-CSMA hybrid, etc. could be flexibly switched via software control for purposes such as controlling the access to a shared medium of communication, energy efficient operation, etc., as necessitated by the prevailing operational and/or service requirements. For example, a CSMA-based star-topology network expending sizable amount of energy could be switched to polling mode so as to mitigate both power and packets loss (suffered as a result of data collision). Similarly, a star network operating in ‘polling’ mode could be configured to operate as per CSMA mode to enhance throughput and make better utilization of (the slots within) the communication medium. Polling and CSMA channel access methods have been imbibed within the Contiki-based software C code fed within the end devices by means of incorporating

an ‘if’ conditional statement within the ‘broadcast_open’ function. When operating in ‘polling’ mode, the specified ‘If’ condition within the code is set to true owing to which each end device transmits packets in their respective set time slots. On the other hand, CSMA mode gets executed independent of this particular ‘If condition’. This has been elaborated by means of pseudo codes along with pertinent explanation below.

The code with which the centrally placed coordinator (node 9) is configured consists of a counter which increments in accordance with value used within its ‘etimer’ function upto the number of leaf nodes within its cluster (eight in this case) before repeating the counting sequence over again. It also broadcasts these counter values as it goes. The psudo code relevant to this is as below.

```
while(1) {

    PROCESS_YIELD();
    if(ev == PROCESS_EVENT_TIMER) {
        leds_on(LED_PERIODIC);
        counter++;
        a[0]=counter;
        a[1]=TIME_IN_SEC - 1;
        a[2]=1;

        etimer_set(&et, CLOCK_SECOND*'N');

        /*Packets are transmitted (broadcasted) using this function*/
        packetbuf_copyfrom(&a, sizeof(a)); // where ‘a’ is an array variable containing the
                                           counter values to be broadcasted as one of its
                                           elements.

        broadcast_send(&bc);

        if(counter==Number of leaf nodes within its cluster)
            {
                counter=(counter % Number of leaf nodes within its cluster);
            }
    }
}
```

Each of the leaf nodes are configured with their own unique address and (respective) node IDs. The codes with which they are configured firstly check if the ‘incoming counter value’ (broadcasted by the coordinator node) matches with their respective node ID. If so, a ‘Transmit flag’ gets set. Only if this ‘Transmit Flag’ is ‘set’ does the broadcast function of the leaf node get executed, allowing it to transmit the data sensed by it over to the coordinator (i.e., an ‘If-conditional’ statement is used to check if the ‘Transmit Flag’ is set or not prior to transmission of the data). In order to flexibly switch over to operating in CSMA mode, the same ‘broadcast section’ of the code is written ‘independently’ or in separation of the ‘If-conditional statement’ involving the ‘Transmit flag’ variable, and can be triggered via an external radio signal message (again via a different ‘If-conditional’ statement). This has been expressed by means of a pseudo code as below:

```

static void
broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    uint16_t *received_node_ID;
    received_node_ID = (uint16_t *)packetbuf_dataptr();

    if(received_node_ID[0] == node_ID_of_receiver_node) // Check if incoming
counter value is the same as node ID
    {
        Transmit_Flag=1;    // Set 'Transmit flag'
        a[0]=dataptr_temp[0]; //Receive polling counter values
        a[1]=dataptr_temp[1];
        a[2]=dataptr_temp[2];
    }
    else
    {
        Transmit_Flag=0;
    }
}
PROCESS_THREAD(cc2538_demo_process, ev, data)
{
    while(1) {
        if(Transmit_Flag==1) //For Polling mode
        {
            packetbuf_copyfrom(&c, sizeof(c)); //where 'c' is array of sensed data
variables to be transmitted
            broadcast_send(&bc);

            Transmit_Flag=0; //Resetting the 'Transmit flag'
        }
    }
}

and
// For 'CSMA i.e., 'data-bursting mode' mode
while(1) { // If Transmit flag =0 or if a requisite external command message
directing it to execute in CSMA mode is received,
{
    packetbuf_copyfrom(&c, sizeof(c));
    broadcast_send(&bc);
}
}

```

Complete codes for both leaf nodes and sink node or coordinator node are placed within section A.3 of the appendix.

With regard to the above MAC layer-based parametric flexibilities, a simulation based experiment was performed on the virtual environment offered by Cooja simulator to evaluate the impact of re-orchestrating the 9-node star topological network (discussed in section 5.2 and depicted in running condition in figure 5-12) from CSMA to polling protocol on the total packet loss suffered by the network. The data rate of each of the end devices in the the 9-node VSC (Virtual Sensor Cloud) network, depicted in running condition in figure 5-12, was set to (a high value of) 200 samples per second for both the experimental cases.

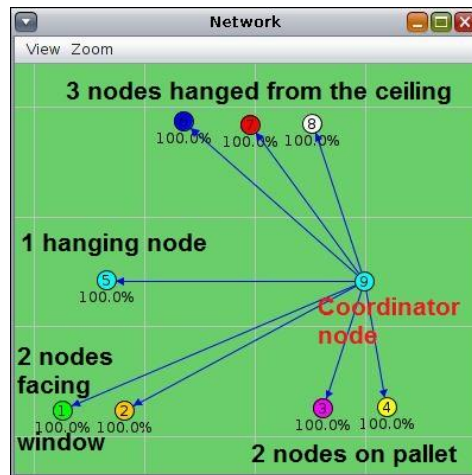


Figure 5-12 9-node virtual network in running condition within the Cooja simulator.

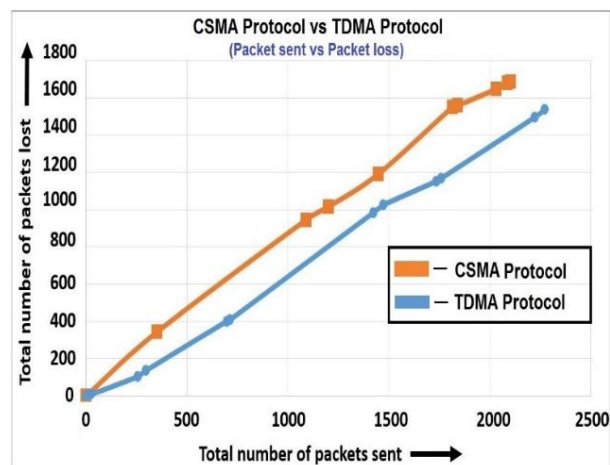


Figure 5-13 Graphical comparison of CSMA and TDMA network protocol instances with respect to packet loss for the virtual 9-node star network implemented within 'Cooja'.

From figure 5-13, it is evident that the packet loss suffered by the network upon implementation of TDMA channel access method is somewhat lesser than when CSMA channel access method is adopted (for the scenario considered). From this, it can be inferred that implementation of TDMA channel access method tends to lead towards somewhat greater network reliability (more so if the data communication rate ranges from low to normal). Moreover, similar flexible node-operational as well as MAC layer parametric manipulations could further mitigate the problem of packet losses, especially in network operations involving communications taking place at a high data rate.

From the above account, it is evident that

- Software-defined switching from CSMA channel access method to TDMA channel access method curtails the overall packet loss suffered by the network to a certain extent. From this, it can be inferred that implementation of TDMA channel access method would lead to somewhat greater network reliability for this network architecture (especially for networks where packet losses are either to be reduced to the extent possible when data communication rate is high or better yet, eliminated, provided high rates of data communication are not required). Moreover, similar functional (node-operational MAC layer parametric, topological, etc.) manipulations could potentially further mitigate the problem of packet losses, especially in network operations involving high data rate transmissions (of the order of 200 samples per second as depicted in the above example). Topological manipulations resulting out of software-defined functional reformulations could alleviate unequalized load distributions as alluded to through the second simulation based example.
- by virtue of a having a virtualization environment in place, the impact of software-defined functional manipulations can be soft-trialled prior to implementation on the physical layer (thereby justifying the proposed ideology).

5.4.8 Channel Allocation

The MAC layer could also be accessed to exert control over allocation of channels to nodes as a means to regulate traffic within the shared medium of communication. Such frequency-based clustering that could result in realizing/creation of multi-channel sensor networks could be accomplished through software reconfiguration of a certain desired group of sensor nodes with a different channel (out of the original single network). Alterations pertaining to channel assignment can be accomplished via alteration of the numeric value associated with the ‘#define CC2538_RF_CONF_CHANNEL’ line within the ‘contiki-conf.h’ header file, to assign the desired channels to the node. For example, in order to configure a TI CC2538 node to transmit (at one of channels available from 11 to 26) at, say, channel 26, the following section of the code would be required within the ‘contiki-conf.h’ file,

```
#define CC2538_RF_CONF_CHANNEL      26
```

Within the Cooja-provisioned virtual environment, channels allocated to the virtual Cooja nodes may be altered through software control via altering the pertinent macro definition ‘RF_Channel’ within the respective header file (i.e., ‘contiki-conf.h’) placed within the Cooja directory within the Contiki.

The incremental example implementations discussed above have been presented with the sole intention of reflecting the role of Contiki-based software control in manipulation of certain basic parameters that could serve as ‘drivers’ for re-orchestrations to be implemented on a wider scale (throughout a particular Contiki-configured sensor network). Such software-controlled manipulations could significantly contribute towards the fluid interaction of the sensor network with the monitored external phenomenon. On a significant note, running soft-trials of the various cluster formation possibilities within the virtual simulator of Cooja present within Contiki can facilitate converging upon the optimal or near optimal cluster formations and network configuration parameters in an expedited manner, without interrupting the PSC data collection process.

5.4.9 Network (Topology)-Operational (Functional) Re-orchestrations

The Contiki software has been used to (pre-)configure physical CC2538 nodes with either leaf, router or gateway functions. However, in order to allow for dynamic software-defined re-orchestration, Contiki-based software control could be used for

- generating and compiling the ‘Contiki-based’ C programs inclusive of all the three viz., leaf, router and gateway functional components
- pre-configuring the network CC2538 nodes with such C codes (comprising of all the three key functionalities, albeit it may only be initialized with the main or dominant functional role whilst others remain dormant.)
- switching to the other (initially dormant) functional modules embedded within a node (by means of selecting the pertinent functional module within the code). This could be realized by means of having in place a conditional statement such as ‘If-Else’, ‘Switch-case’, etc., which could be triggered by means of external impulse radio signal message whenever required.
- activating multiple functional modules simultaneously.

A generic schematic reflecting the various possible functional roles that could be undertaken by CC2538 nodes when pre-configured (i.e., loaded) with the Contiki-based ‘C’ functional modules catering for those functionalities, is as depicted in figure 5-14.

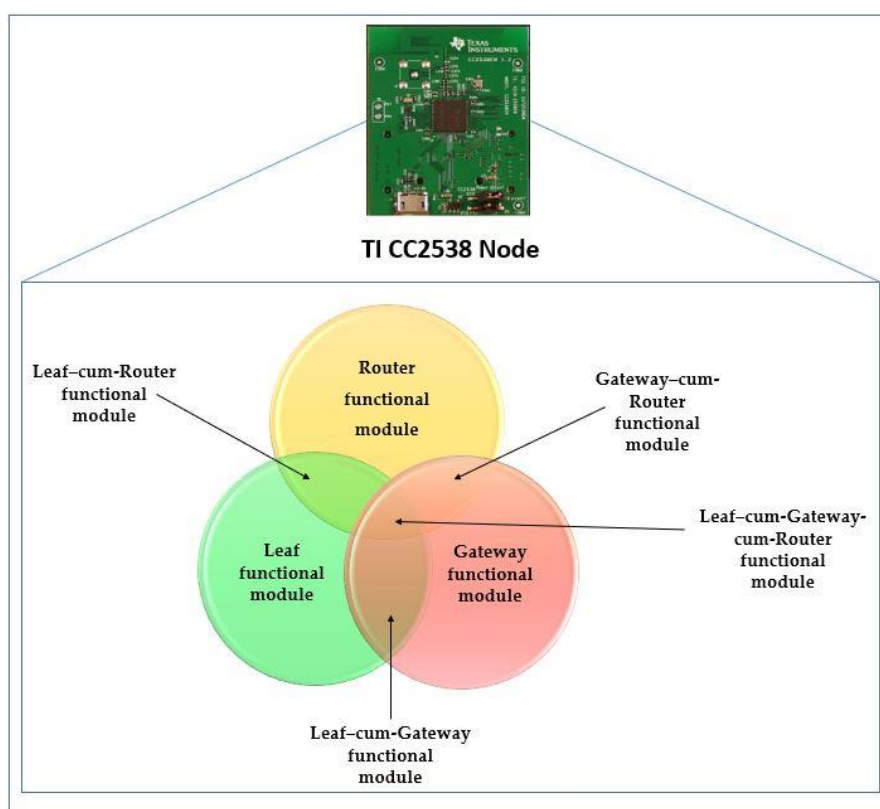


Figure 5-14 Schematic representing the various core and multi-functional capabilities that can be assumed by the TI CC2538 device owing to Contiki-based software control.

The above account is key to how Contiki Software control forms the basis for flexible switching or augmenting the functional capabilities of a multi-functional capable device (such as the TI CC2538).

The flexibility of selecting and activating a particular function offered by the Contiki software, including the ability to execute multi-functional roles simultaneously (provided the hardware is capable of accommodating for that multiple such functions) opens the door for a plethora of network re-orchestrations, thereby vastly augmenting the degree of freedom.

As alluded to in chapter 3, the in-built simulation platform of Cooja within Contiki allows for virtualization of such physical implementations and soft trialling of various re-orchestration scenarios. This is in keeping with the ideology of SDN wherein the cloud layer (acting as the control plane) provisioning for,

- virtualizing the functionalities (i.e., decoupling them from the underlying physical data plane) and
- exertion of software control over such abstracted form of these functionalities, i.e., virtual functions (so as to determine suitable re-orchestrations beforehand)

forms the basis of organising (remote and dynamic) flexible control over the physical layer.

5.5 Example WSN System Implementation

As a means to reflect our proposed ideology of a software-defined sensor network operating under the aegis of a cloud-based organization allowing for virtualization and soft re-orchestration of the underlying (physical) layer, a hybrid sensor network system i.e., one consisting of both physical and virtual environments has been implemented (within our laboratory premises), as already alluded to in section 3.3 of chapter 3.

5.5.1 Physical Implementation

Figure 5-15 [41-42][50] depicts a 9-node Texas Instruments (Contiki-ported) CC2538 SoC-based (physical) sensor network implemented within our lab premises[41-42,50]. Herein, eight of the nine wireless CC2538 sensor-transceivers were configured using Contiki to act as end devices capturing ambient light, temperature and radio signal strength data in their respective timeslots and reporting them to a centrally placed IoT-based CC2538-cum-Raspberry Pi unit. By means of a python script, Raspberry Pi escalates the sensed data so received from the end devices, to our local server, over the Internet[41-42, 50].

As depicted in figure 5-15 [41-42][50] above, nodes ‘1’ and ‘2’ were placed near the window to facing outside (the lab) so as to capture the ambient light during the day, i.e., incident sunlight falling on it. Nodes ‘6’, ‘7’ and ‘8’ were hung towards one side of the room from the lab ceiling. Node ‘5’ too, was hung from the lab ceiling at one particular area of the lab as depicted in figure 5-15 [41-42][50] above. Nodes ‘3’ and ‘4’ were deployed against an empty wooden pallet structure present at one particular location in the room, as shown in figure 5-15 [41-42][50].

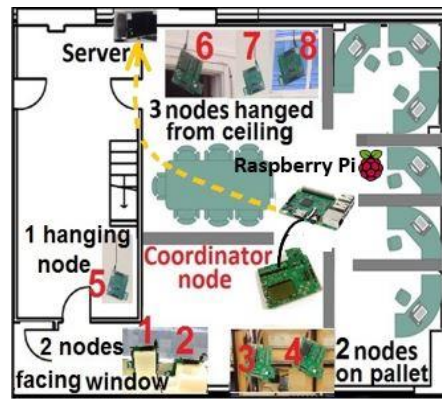


Figure 5-15 Physical implementation of a 9-node TI CC2538-based sensor network within our laboratory premises.

The sensing requirements of the physical sensor network so established within the lab premises merely involved monitoring of the indoor environmental parameters of ambient light and temperature, along with the radio signal strength (roughly indicating human presence and movement). As already alluded to in section 3.3.2 within chapter 3, the graphical representation of the stored data (so collected from the physical network) provisioned via our server database is as shown in figure 5-16 [41-42][50].

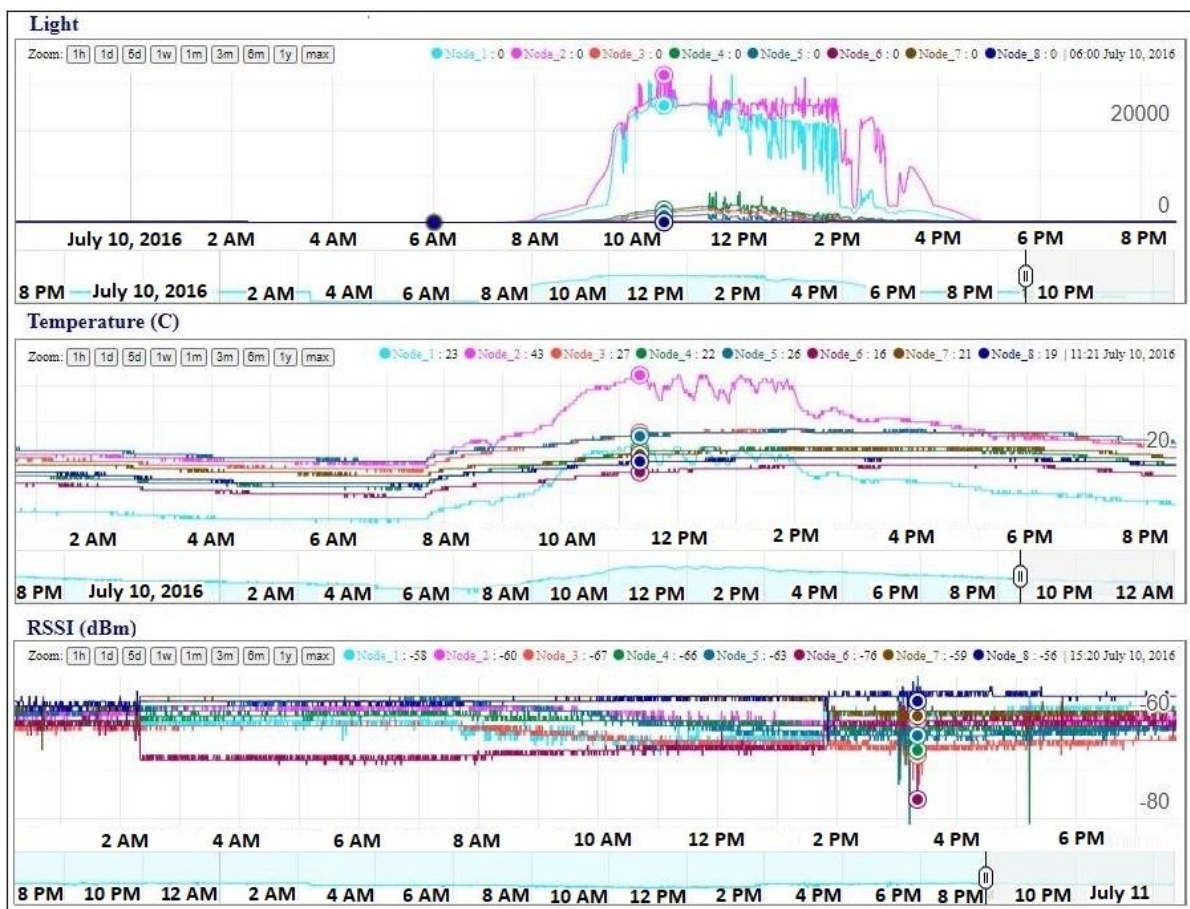


Figure 5-16 Graphical trend of 'sensor' data (ambient light, temperature and RSSI) captured by the 8 leaf nodes, retrieved from server database.

As is evident from both figures 5-15 as well as 5-16 [41-42][50], nodes ‘1’ and ‘2’ placed near the window facing direct sunlight during daytime reflect vastly higher ambient light sensor readings as compared to the other 6 leaf nodes. Furthermore, the same two nodes report somewhat higher temperature values as well during the same daytime period, as compared to the other leaf sensor nodes.

It is deemed worthwhile to re-assert (via repetition of the figure 3-5 in figure 5-16 within this section) that even under normal (i.e., routine or predictable) circumstances, considerable variations of the sensed values are observed (within a span of two days). The intent This necessitates establishment of sensor network system endowed with requisite operational flexibility to cope and efficiently capture the significant dynamics of monitored environmental phenomenon, if it were to occur.

5.5.2 Implementation of the Virtual Environment Within the Remote Server

As alluded to in chapter 3, Cooja-based virtualization allows for the precise replication of the logical operations occurring within the physical network. It is, however, not possible to replicate the physical data sensed by the (corresponding) real-life sensor nodes (without external, typically cloud-provisioned, support). Moreover, it does not provision for modelling of the physical environment. Owing to such inherent limitations, soft-trials conducted on such a platform may result in significant deviation from a desirable accurate outcome. As a step towards alleviating this deficiency to a certain extent and attain a somewhat more realistic representation of the dynamics transpiring within the physical WSN, a process facilitating for incorporation the of element of reality within such a virtual platform in ‘real time’ was devised, as explained below.

Real-world data acquired from the Raspberry Pi gateway over the Internet is stored within the data repository within the server. By means of the relevant database interface, data from the most recent physical run (from the database) is extracted and fed into the Contiki configuration unit. Equipped with such real-time ‘sensed data’ information, Contiki ensues upon necessary compilation and configuration process to create ‘virtual Cooja mote’ copies within the Cooja WSN simulation platform resulting in virtualization of the physical sensor network. A generic model reflecting the practicable implementation undertaken towards the same is as depicted in figure 5-17.

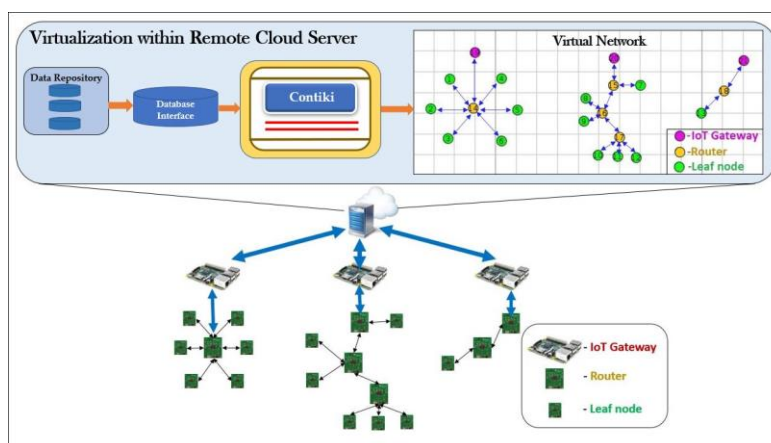


Figure 5-17 Overview of the setup devised towards facilitating for incorporation of the element of reality within the cloud-based virtualization platform in real time.

As a means to further elaborate upon the above, a more detailed block diagram is presented in figure 5-18 [41][50] and discussed as below.

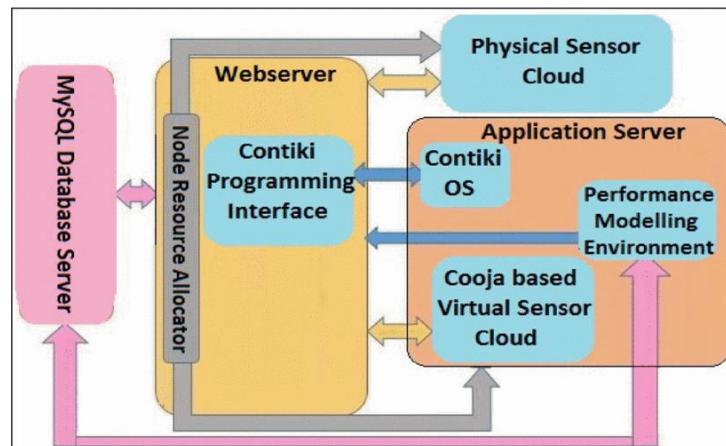


Figure 5-18 Block diagram depicting Remote server implementation and the inter-relation among its various components.

The remote server implemented for data storage, data visualization, configuration, etc. purposes has three main components associated with it namely, the ‘MySQL Database Server’, the ‘Webservice’ and lastly, the ‘Application Server’[50]. The Application Server hosts the sub-components of ‘Contiki OS’, ‘Performance Modelling Environment’ as well as the Cooja-based virtualization environment. The sub-component of Contiki IDE serves to configure both the physical and virtual sensor networks. Tools such as MATLAB hosted by the ‘Performance Modelling Environment’ serve to evaluate the performance of the network, optimizations obtained from which are directly fed to the ‘Contiki Configuration Interface’. The Contiki Configuration Interface utilizes these suitable re-orchestrations to re-orchestrate both physical and virtual networks. Lastly, the function of the ‘MySQL Database Server’ is to write queries to the database. It does so by means of a PHP script which (also) fetches data obtained from the physical leaf nodes. This incoming data gets timestamped and stored in rows.

Certain specificities pertaining to the server implementation are provided below[50]:

- To start with, sensor data emanating from the physical leaf nodes are received by the IoT Coordinator i.e., the Raspberry Pi as depicted in figure 5-15 [41-42][50], which in turn is escalated to the webservice which serves as the entry point for all the incoming sensor data. It does so by using REST API. Usage of REST API serves to exchange the necessary information between the database and the application server, besides fulfilling the important requirement of communicating with users. By means of GET and POST commands, users select the nodes they wish to view the data for. The client side consists of the python script residing within the Raspberry Pi which pushes the data over to the Webservice. This data gets forwarded over to the MySQL database (via REST APIs) for statistical and data processing purposes pertaining to each of the individual CC2538-based node. For scripting purposes, the PHP script is employed by the server.
- Optimizations foreseen within the ‘Performance Modelling Environment’ serve as inputs to the Webservice wherein the resident Contiki Configuration Interface performs the function of implementing them on to both virtual and physical nodes.

5.6 Example WSN Re-orchestration Scenarios

Implementations of certain example re-orchestration scenarios have been realized via exerting Contiki-based software control over the three ‘core’ network functionalities i.e., Leaf, Router and Gateway functions. Such re-orchestrations have resulted in changes pertaining to network topology, network dataflow (direction), performance, etc., besides serving as a means to clearly define the re-orchestration phase (as well as the phases preceding it) and analyse its implications on the network, in terms of network downtime.

5.6.1 Simple Network (Function) Manipulation: Function Swapping Case

Consider the following simplistic case of network re-orchestration wherein Contiki-based software control has been employed to

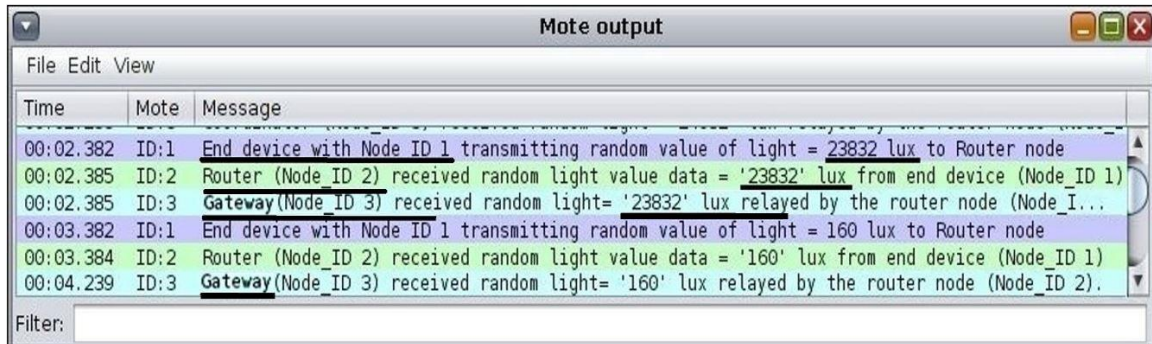
- (initially) pre-configure three nodes i.e., node ID: 1, node ID: 2 and node ID: 3 within Cooja to execute the ‘leaf’, ‘router’ and ‘gateway’ functionalities, respectively, constituting a three-node ‘multi-hop network’, as shown in figure 5-19a [26].
- manipulate to functional roles being executed by two of the three nodes, i.e., node ID: 2 and node ID: 3, as depicted in figure 5-20a [26].

Figure 5-19 a [26] illustrates the initial network setup wherein the flow of data is such that the data sensed by the virtual node (with Node ID: 1) configured with the leaf function (i.e., light, temperature and RSSI values as sensed by its physical counterpart) is relayed to the virtual node (with Node ID: 3) configured with the Gateway function, via the intermediate virtual node (with Node ID: 2) configured with the Router function. The ‘mote output’ window within Cooja for this initial network configuration is as depicted in figure 5-19 b [26].

Upon modifying within the ‘broadcast_open’ function within Contiki-based firmware to enable the desired (pre-existing but initially dormant) functional module(s) (and de-activating the existing functional module), a separate Contiki functional module is generated, which when implemented onto the same virtual node results in re-orchestration of the operational behavior of the same node (in this case, re-orchestration of a virtual node to a router role from an end device role and vice-versa). Such individual node re-orchestrations tend to contribute to the re-orchestration of the entire network as a whole, including its inherent dataflow. This aspect is elaborated below.



(a)



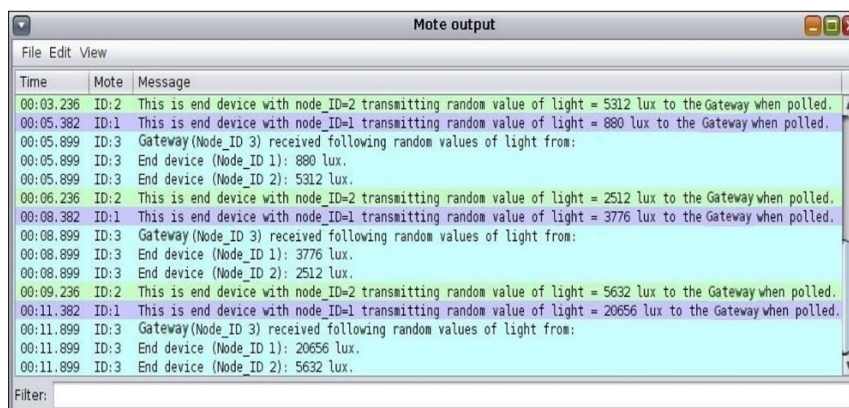
(b)

Figure 5-19 (a) Virtual three-node multi-hop network implemented within Cooja; (b) ‘Mote output’ window within Cooja reflecting the dataflow within the multi-hop network.

As depicted in figure 5-20 a [26], activation and due implementation of the (relevant and initially dormant) ‘leaf’ functional module within the pre-existing functional code fed to node ID: 2 (initially configured to act as a router) through Contiki-provisioned software control re-orchestrates its functional role to that of a leaf node.



(a)



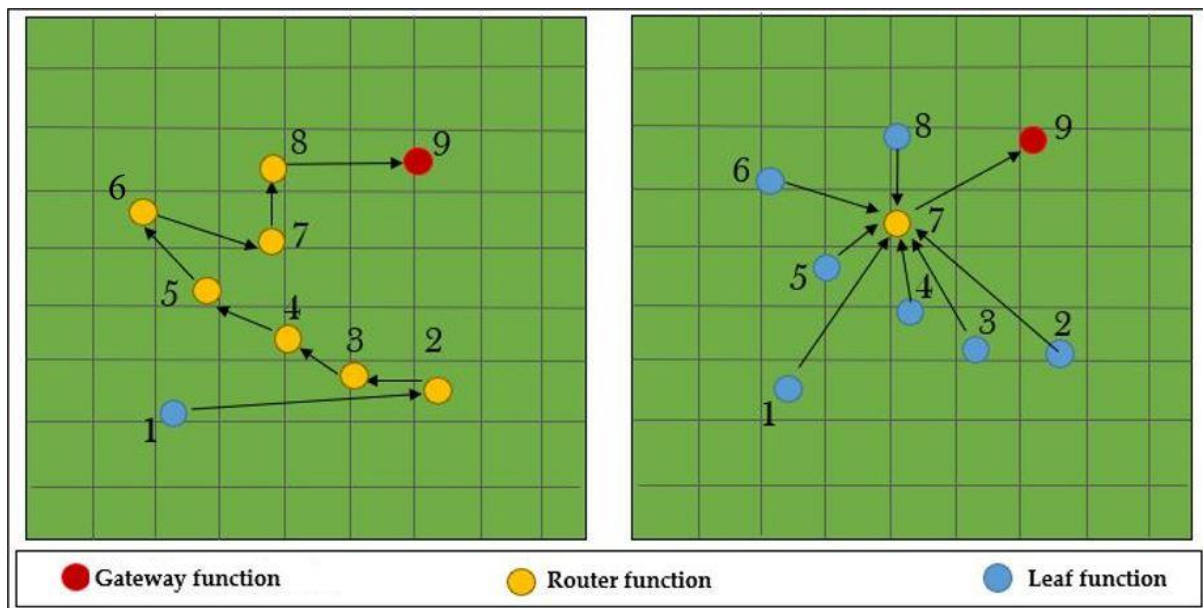
(b)

Figure 5-20 (a) Star topology post re-orchestration of the multi-hop network (depicted in Fig. 4); (b) Star topology behaviour of the re-orchestrated network depicted by the ‘Mote output’ window within Cooja.

This, coupled with re-orchestrating the operation of the gateway node (i.e., Node ID: 3) via similar software control to enable polling of its two end devices results in a star-based topological organization wherein the flow of data is such that the data sensed by end devices (Node ID: 1 and Node ID: 2) is received by the Gateway node acting as sink for the two nodes. The mote output screenshot pertaining to this re-orchestrated network is as depicted in figure 5-20 b [26].

Example cases of software-defined re-orchestrations such as the above could be instrumental in resolving cases of network fragmentation caused by departure of a mobile (router) node beyond the communication range of the Gateway.

5.6.2 Demand for Flexibility: Topology Related-Case



(a)

(b)

Figure 5-21 Certain topological orientations that a given IoT-enabled sensor network could flexibly re-orchestrate to as a result of software-defined re-orchestration:- (a) Multi-hop Topology and (b) Star Topology.

The network is initially configured to operate under a multi-hop topological arrangement, as shown in figure 5-21 a [26]. Herein, the sampling rate or reporting time interval of each of the nodes is gradually increased from one to twenty-five samples per second (increasing 5 samples per second at a time). Overall packets lost by the network (for these sampling rates) has been focussed upon as a performance measure herein. Subsequently, via software control, it is re-orchestrated to operate as a star-topological network as shown in figure 5-21 b [26].

The data communication rate of the data being sensed by the end device is incremented (in steps of five samples per second, from '1' sample per second to 25 samples per second) for both the sensor network topological arrangements in a bid to observe the implications. The packets lost (parameter) has been considered as the performance measure within this experiment.

Table 5-2 The impact of increasing data communication rates on the packet loss experienced by the network consisting of eight nodes for various scenarios.

Data Communication rate Packet/Sec PPS	Packets lost (‘Multi-hop’)	Packets lost (Star-CSMA)	Packets lost (Star-TDMA)
1	0	0	0
5	3	0	0
10	5	0	0
15	8	0	0
20	10	4	0
25	12	--	0

Table 5-2 [26] clearly depicts the considerable mitigation of packet loss experienced by the sensor network when re-orchestrated from multi-hop to a CSMA-based star-topological arrangement, as shown in figure 5-21 b while the network experiences loss of twelve data packets when run on multi-hop topological configuration at 25 samples per second. The Cooja simulator ceases to offer any data when the same network is run on CSMA-based star topological configuration at the same 25 PPS. On switching to a TDMA-based star topology, no packet losses are observed (at least till the sampling rate is increased to 25 samples per second). This experiment aptly demonstrates the benefit of incorporating a virtual environment to test and foresee the implications of software manipulation of individual network functions on the overall network performance.

5.7 Example Network Re-orchestration Scenarios with Focus on Re-orchestration Latency

5.7.1 Simple Network Manipulation: Function swapping case

Although software-defined re-orchestration allows for flexible node-operational manipulations and thereby, if necessary, topological reorganizations, it might be accompanied by service disruption issues. Service disruption refers to the latency associated with the re-orchestration process, and may temporarily render the network partially (or fully) disconnected. It is, therefore, of relevant research interest to explore the impact of the reorchestration process on service delivery. In order to preliminarily investigate the overall latency associated with the disruption suffered by the network, a simplistic experiment involving ‘function-swapping’ among two network nodes (belonging to a three-node network) has been again performed on the virtual environment offered by Cooja.

Herein, a three node stationary network, consisting of end device, router and Gateway node, is considered, as shown in figure 5-23 a. The leaf node continuously transmits sensed data (temperature and RSSI) to the ‘router’ node, which relays this data over to the Gateway node, along with its battery level. These are indicated by messages 1 i.e., $M_{LR_Initial}$ and 2 i.e.,

$M_{RG_Initial}$ respectively, as depicted in figure 5-22. Owing to the continuous routing operation and prohibition from entering ‘sleep’ mode, the battery level of the router is bound to deplete rapidly. It is also assumed that the Gateway node is programmed to initiate the swapping process once the battery level of the router node falls below the set critical threshold, with that of the leaf node. Such a simplistic case of automated, query-based software-defined re-orchestration aids extension the overall network lifetime, to a certain extent. It also serves a means to investigate the latency associated with the reorchestration process, at an elementary level.

The simplistic function swap process of network reorchestration involves certain communication transactions (amongst the three node functions) initiated by the Gateway node as depicted in figure 5-22. Description pertaining to the ensuing communication messages are provided as below.

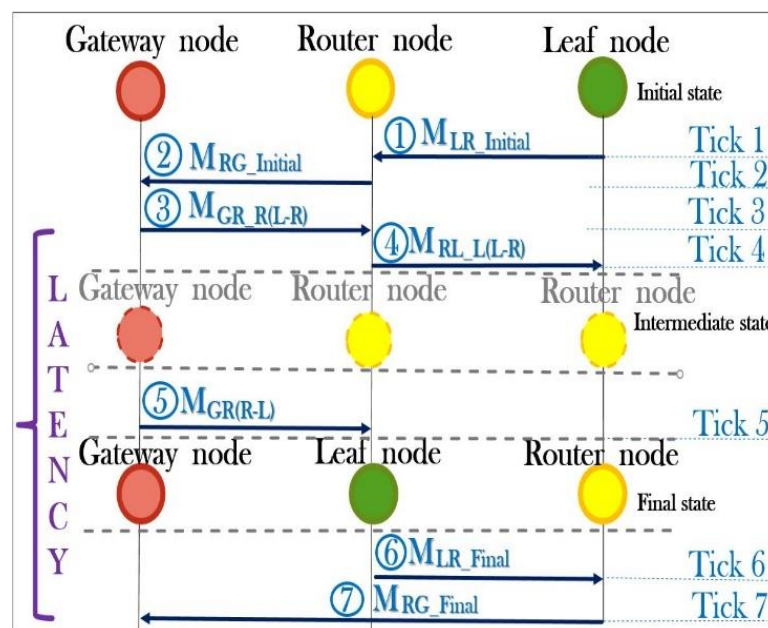


Figure 5-22 Communication messages exchanged amongst the constituent network elements to fulfil the desired re-orchestration process of ‘function swap’ process between the leaf and router nodes.

1. Upon detecting the fall of the router node’s below certain pre-defined threshold level, The IoT based gateway node transmits message 3 i.e., $M_{GR_R(L-R)}$ to the router node, which, in turn, transmits message 4 i.e., $M_{RL_L(L-R)}$ to the leaf node, notifying it to turn into a router node. By means of invoking the routing-centric software functional modules (and masking the previously defined leaf-centric modules), the reception of these messages prompts autonomous transformation to a router node.

2. The Gateway node then issues message 5 i.e., $M_{GR(R-L)}$ to the leaf node so as to invoke leaf-centric software functional modules whilst disabling its routing-centric modules. Thus, the router node now turns into a leaf node, relieving it from its routing operational requirements and thus, conserving battery power, to some extent.

3. Owing the swapping of amongst the leaf and router nodes, the dataflow within the network is altered as depicted in figure 5-23b.

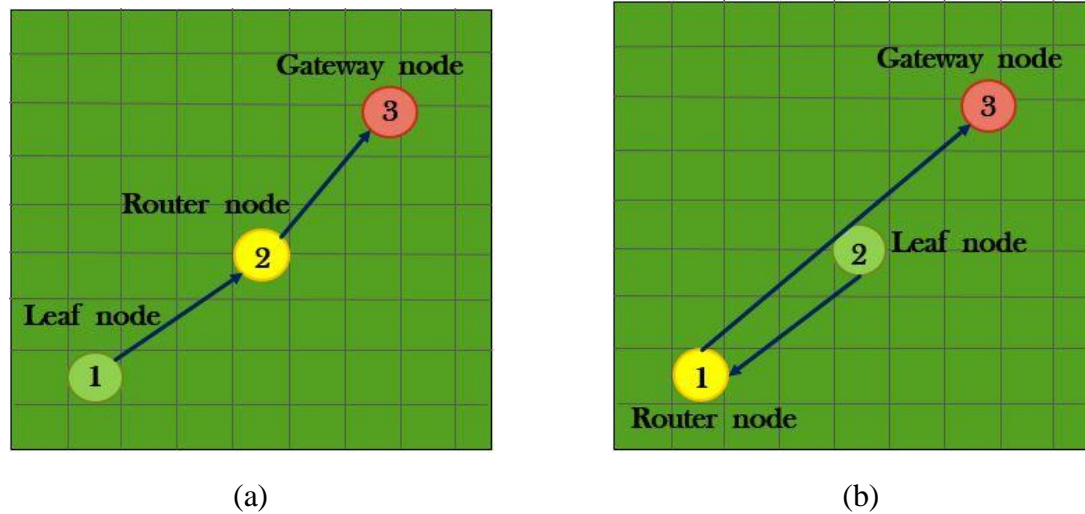


Figure 5-23 Initial state of the three-node network wherein nodes 1, 2 and 3 are pre-configured to behave as leaf node, router node and Gateway node respectively; b) Final state of the three-node network wherein the network has undergone re-orchestration owing to swapping of functions amongst node 1 (now a router node) and node '2' (now a 'leaf' node) causing the dataflow within the network to get altered.

Normal operation resumes in this re-orchestrated network as the router-turned leaf node transmits its sensed data (message 6 'M_{LR_FINAL}') over to its leaf-turned router counterpart, which in turn routes this data over to the Gateway node (message M_{RG_FINAL}), as depicted in figure 5-23 b. Upon summation of time intervals of each of the individual message 'ticks' (obtained through the timestamps seen from the Cooja simulation), the overall 'service disruption time' or 'latency' incurred during the re-orchestration process until normal dataflow resumes within the network was found to be 2.56 seconds. This disruption time is considerable (even in the case of this simple 3-node network), given the individual tick messages for an unperturbed network are of the order of certain milliseconds each. This is illustrated in figure 5-24 as shown below.

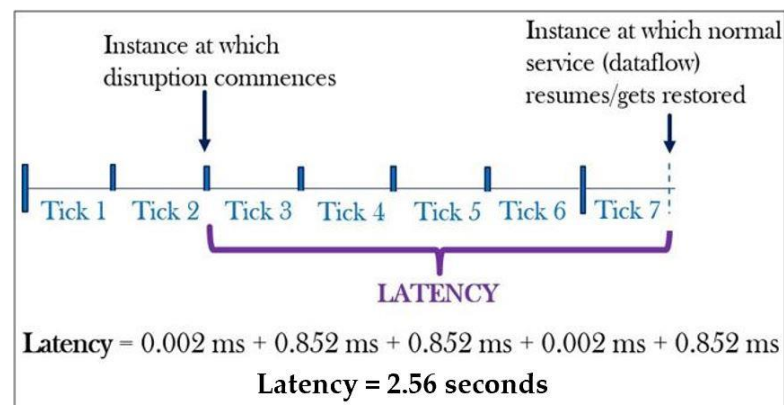


Figure 5-24 Overall latency (associated with the function-swapping-based re-orchestration process) deduced through summation of time intervals of the individual message ticks.

5.7.2 Network Manipulation: Router Replacement Case

As a means to gauge the extent of the downtime experienced in case of a slightly more involved scenario, an example cloud-governed sensor network system due to undergo the process of phased dynamic re-orchestration (as documented in section 4.7.2 within chapter 4) as a result of an impending network fragmentation event has been considered [26]. The virtual representation of this network running within the Cooja simulator has been depicted in figure 5-25 [26] wherein as a result of a certain special circumstance, the mobile router-cum-clusterhead node i.e., node 5, tends to irreversibly drift away from the Gateway node (i.e., node 6). As a means to pre-empt an eventuality wherein the flow of ‘sensed data’ emanating from the ‘leaf nodes’ (represented by the nodes 1, 2, 3 and 4) over to the Gateway node (through the inter-mediate router node in a multi-hop fashion) ceases (as a result of such fragmentation scenario), it is required that the ensuing phased re-orchestration process concludes with the election of the most suitable of the four leaf nodes (which are capable of assuming the role or function of a router) to switch to a role of that of a ‘replacement’ router, before the ‘departing’ router ventures out of the connectivity range. For simplicity’s sake, it has been assumed that the leaf nodes cannot transmit sensed data over to the Gateway node, even if it happens to be within (any of) their communication range, but only through an intermediate router node. However, if need be, the Gateway node is fully capable of establishing direct connectivity with any of the leaf nodes.

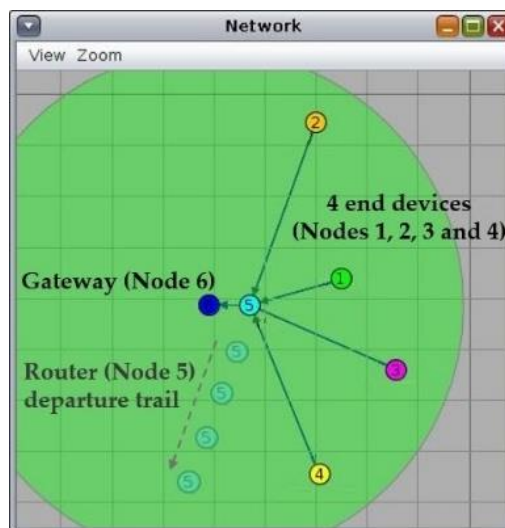


Figure 5-25 Cooja-based virtual representation of a 6-node network facing impending network fragmentation owing to departure of a mobile router node away from range of connectivity [26].

During the first phase of ‘Data Analysis and Event-Identification’ phase’, a particular dedicated knowledge component hosted within the ‘Data and Knowledge’ repository continually monitors the strength of radio signal messages exchanged between the Gateway node (i.e., node 6) and the router node (i.e., node 5). By means of continuously analysing the historical data of the strengths of the radio signals so exchanged between the gateway and the router nodes (example representation of which has been presented in Table 5-3), it identifies the pattern of departure of the router and subsequently ensues upon triggering an alert to proactively initiate the ‘Re-orchestration Planning’ phase.

Table 5-3 Example representation of mobile router communication strength.

Time instants	RSSI (dBm)
m	-13
m+1	-16
m+2	-27
m+3	-44
m+4	-61
m+5	-77
m+6	-94

Within the ‘Re-orchestration Planning’ phase, another dedicated knowledge component ensues upon figuring out the most suitable leaf node that can replace the departing router. For the example network in consideration, it has been assumed that only leaf nodes with IDs 1, 2 and 3 are capable of assuming router functionality, barring the leaf node with node ID 4 from being a part of the election process (refer to figures 5-25 [26] and 5-26 [26]). The election process involves computation of a fitness model for each of the three participant nodes as a means to determine (and subsequently compare) their normalized weights. The three factors that have been identified as the key requisite parameters for the fitness model in consideration are as follows:

- Radio signal strength of each of the participant leaf nodes with respect to its counterpart (both participant and non-participant) leaf nodes.
- Radio signal strength of each of the participant leaf nodes with respect to the gateway node.
- The battery power level of each of the participant leaf nodes.

In this particular case, each of the above three parameters have been assigned equal weightage so as to enable the dedicated knowledge component to compute the normalized weights of each of the participant leaf nodes. Such assumptions (pertaining to the assignment of weightages for the different parameters) however, are subject to change, depending either upon the case for which they are being stated, or as per the knowledge derived through a process of long-term learning.

The fitness model (encompassing the aforementioned three parameters) is expressed mathematically as below:

$$W_{NW} = [a_j \times \text{RSSI}_{EDs_AVG}] + [a_{j+1} \times \text{RSSI}_{G-ED}] + [a_{j+2} \times \text{BP}_{EDs}] \quad [26],$$

where,

$a_j = L \times (M_j)$, $a_{j+1} = L \times (M_{j+1})$ and $a_{j+2} = L \times (M_{j+2})$ denote the respective equalized or normalized weightages assigned for each of the three parameters (where M_j , M_{j+1} and M_{j+2} represent the fiddle factor associated with each parameter),

‘ RSSI_{EDs_AVG} ’ denotes the average of the RSSI values for any given participant node with respect to its counterpart (both participant and non-participant) leaf nodes,

‘ RSSI_{G-ED} ’ denotes the RSSI value for any given participant leaf node with respect to the ‘Gateway’ and

‘ BP_{EDs} ’ denotes to the instantaneous ‘battery’ power ‘level’ of a given ‘participant’ leaf node.

Herein, the ‘fiddle’ factors associated with each of the parameters are considered (or formulated) in such a way that they are completely independent of each other. The intention behind this approach was to ensure that equal (and uniform) weightages are assigned to each and every variable. Weightages assigned to each of the parameter, however, may be skewed in favour of a parameter that may deserve more weightage as compared to other (A more dominant parameter may be identified through experience of the application being dealt with). As a means to elaborate upon the fitness model so formulated, a sample router fitness calculator table for a given participant node towards electing replacement router (to replace the departing router node) has been provided in section A.4 in the appendix. Figure 5-26 [26] presents an abstracted communication sequence diagram representing the various messages (in the order of occurrence) being exchanged amongst the various nodes over the three phases of re-orchestration. A more elaborate communication sequence diagram showing all the messages transpiring amongst the various nodes over the three phases of re-orchestration process, along with the associated description, has been included in section A.5 of the appendix.

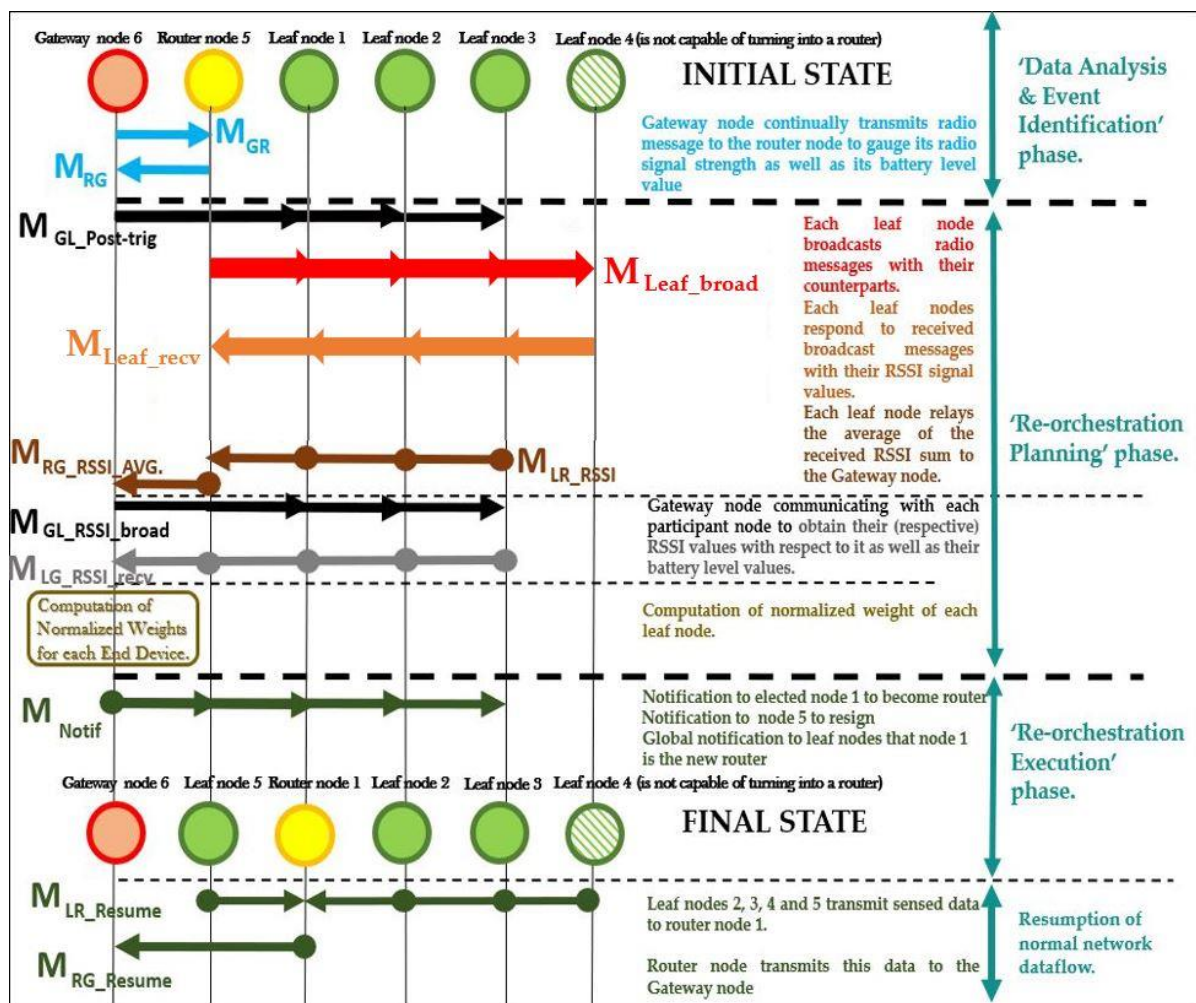


Figure 5-26 Abstracted sequence diagram showing the messages transpiring over the three re-orchestration phases in pursuit of electing the most suitable leaf node to take up the role of the replacement router.

As can be seen from figure 5-26, the ‘Re-orchestration Planning’ phase entails numerous exchanges of communication messages amongst all the constituent network nodes so as gather the requisite information for the computation of the fitness models for each of the nodes and reaches its conclusion upon figuring out the leaf node with the highest fitness. The final phase

of ‘Re-orchestration Execution’ commences with a series of notifications transmitted by the Gateway node which include notifying the leaf node with the highest fitness value of its new ‘role’ as a ‘replacement router’, notifying all the nodes regarding the outcome of the election process (i.e., the elected leaf node would act as the new router-cluster-head for them), directing the departing router to relieve itself of its role as a router, if need be, etc. It concludes with all the constituent nodes of the network in consideration operating in accordance with their respective post-re-orchestration’ roles, thereby resuming the flow of data within the network.

At the initial stage (refer figure 5-26), node 6 (i.e., the gateway node) continuously transmits message ‘M_{GR}’ to node 5 (i.e., the router node) to gauge its radio signal strength and battery level as part of the ongoing ‘Data Analysis and Event Identification’ phase. In response, the router node transmits this information to gateway node 6 via message ‘M_{RG}’. This data gets escalated and stored within the cloud where it is monitored by the requisite knowledge component. The second phase of ‘Re-orchestration Planning’ phase gets initiated by the message ‘M_{GL_Post-trig}’ broadcasted by node 6 to all the leaf-function nodes i.e., node 1, node 2 and node 3 (once it is detected and ascertained that the departing router is set to move out of the network’s connectivity chain). This message directs node 1, node 2 and node 3 to switch over to the router-functional role, if they are capable of turning into router nodes. These nodes then broadcast a series of messages denoted by to each other (including leaf node 4 that is incapable of transforming into a ‘router’) to obtain their radio signal values. The messages are responded to by the leaf-turned router nodes to the other leaf-turned router nodes with their resp. radio signal strength values, as denoted by message M_{Leaf_recv}. Each leaf node then transmits message ‘M_{LR_RSSI}’ (consisting of the averages of their resp. radio signal strength values with respect to their counterpart leaf nodes) to the departing (node 5) router, which in turn is relayed by node 5 to (gateway) node 6 via message ‘M_{RG_RSSI_AVG}’ (consisting of the combined information pertaining to the RSSI values).

Subsequently, messages ‘M_{GL_RSSI_broad}’ is broadcasted by node 6 to nodes 1, 2 and 3 to fetch their radio signal strength with respect to itself as well as their resp. battery values. Message ‘M_{LG_RSSI_recv}’ represents the responses transmitted by all the leaf nodes to the gateway node. Equipped with all the requisite information, the gateway node 6 ensues upon computation of normalized weight of each of the participant leaf nodes (in accordance with the fitness model explained and specified earlier). Upon receiving all the parameters from the different participating nodes in the election process, the relevant knowledge component ensues upon computation and comparison of the normalized weights of all the nodes participating in the router election process. The leaf node with the greatest normalized weight value of all the participant nodes receives a message from the gateway node directing it to assume the role of replacement router. A screenshot of the mote output window, (a feature available) within Cooja reflecting the outcome of the election process (computed within the gateway i.e., node 6) is depicted in figure 5-27.

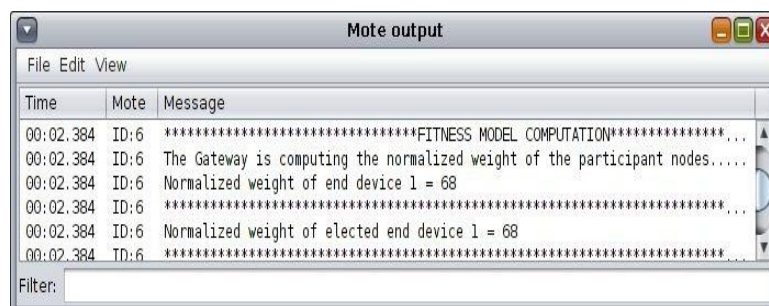


Figure 5-27 Messages related to the outcome of the election process as seen within Cooja’s ‘Mote output’ window [26].

The ‘Re-orchestration Execution phase’ is the final phase wherein a series of messages denoted message ‘M_{Notif}’ (in figure 5-26) get transmitted by node 6, notifying node 1 to turn into a router, directing node 5 to resign from its role of a router-cum-clusterhead and informing rest of the (participant as well as non-participant) nodes about the outcome of the election (i.e., node 1 is the new router node) and reverting back to their original roles as leaf nodes. Finally, normal flow of data resumes within the network wherein all the leaf nodes transmit their data to the newly elected router node 1, as represented by the combined message ‘M_{LR_Resume}’ (also involving node 5 as well once it traverses back to a position that falls within the communication range of the gateway), and the newly elected router node, in turn relaying this information to node 6 via message ‘M_{RG_Resume}’.

The screenshot of Cooja’s ‘mote output’ window shown in figure 5-28 denotes the instant of time of node 1’s functional switching from the role of a leaf node to that of a router node 1 and subsequently commencing upon its duty of gathering sensed data from the leaf node devices.

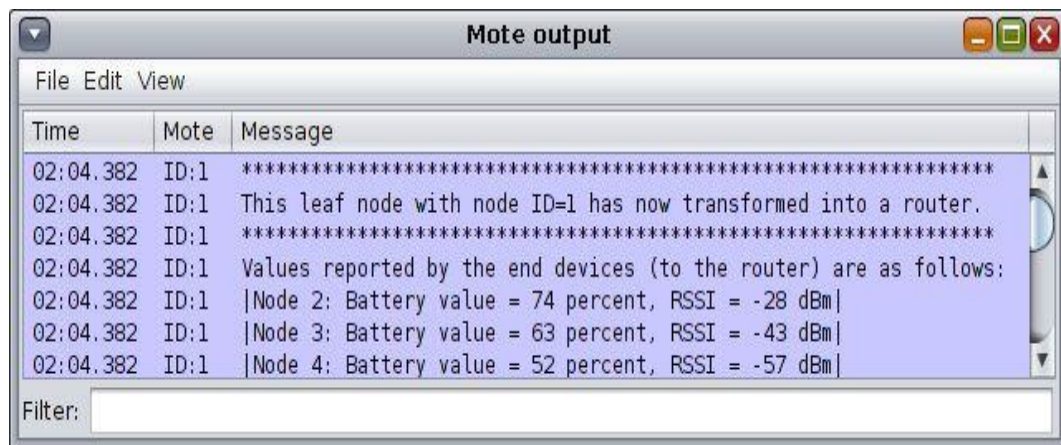


Figure 5-28 Screenshot of Cooja’s ‘Mote Output’ showing the time instant of node 1’s functional transformation from the role of a leaf node to that of a router node and subsequent data acquisition from its constituent leaf nodes [26].

For scenarios where the implementation outcomes are pre-emptively derived (i.e., before the flow of data within the network gets disrupted as a result of network fragmentation), this phase accounts for the actual network downtime experienced by the physical network within the physical layer. For this particular example scenario of network re-orchestration, the downtime experienced by the network equals the time consumed for a total of six messages to get ‘exchanged’ amongst the relevant constituent nodes. The overall ‘re-orchestration’ latency, on the other hand, takes into account the latencies incurred during each of the three re-orchestration phases (i.e., the summation of latencies incurred during all of the three phases).

It is, however, important to acknowledge that results such as the one obtained above are relative in nature. More exact values of network downtimes can be extracted by taking into account the real-world factors that accompany the communication processes, in addition to the node network-level operational parameters viz., sampling rate, number of nodes, topological orientation, etc.

The above example, however, brings to the fore the significance of having in place a cloud layer (consisting of a virtual platform and software knowledge components) for pre-emptively deriving suitable re-orchestrations to be applied onto the physical network. The software knowledge components could, however, also reside at the devices capable of executing ‘edge computing’-based activities i.e., the router and gateway nodes (facilitated by softwarization).

In order to include the element of reality in one of the example processes depicted, the physical RSSI has also been characterized (with respect to increasing) (for the given environment) and used within the virtualization process to reflect the mobility dynamics for the given virtual process. This reflects the saliency of the virtualization platform in accommodating for real data emanated from the physical network towards offering accurate control information for manipulating it.

As part of the physical implementation pertaining to the above example, a 6-node network has been implemented within our AUT-SeNSE lab as depicted in figure 5-29. Here, Raspberry Pi has been employed as the Gateway node whereas the router node and end device nodes are comprised of the Texas Instruments CC2538 node. Data gathered from the end devices is escalated by the Raspberry Pi (acting as protocol converter) to our remote local server over the Internet. Via the data visualization service provisioned by the cloud layer, a graphical representation of this data can be viewed online.

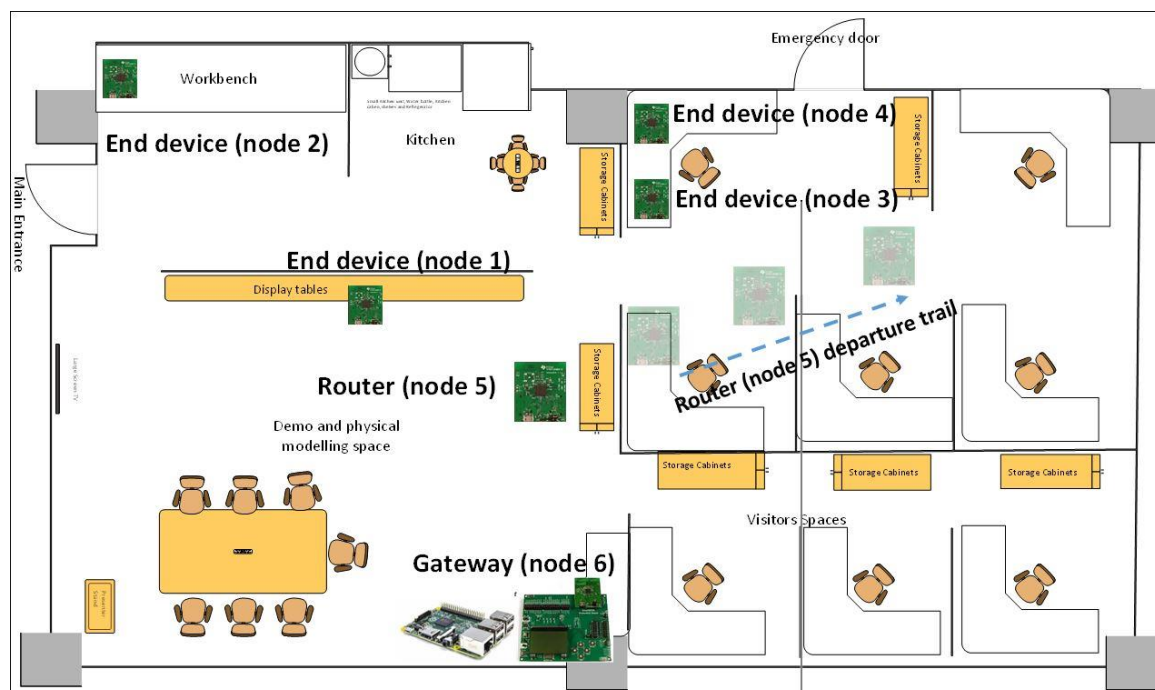


Figure 5-29 Physical implementation of the multi-hop network (corresponding to figure 9) established within the SeNSE lab.

Being continually fed with real-time data or information pertaining to the of the underlying physical network, any change in the topological status (including aspects such as connectivity pattern, data flow, etc.) or operational dynamics of the physical network is reflected within the virtual network. As the router (node 5) moves away from the Gateway (node 6) the radio signal strength value between the departing router i.e., node 5 and the gateway i.e., node 6 gradually

decreases. Herein, a particular dedicated knowledge component (which is a part of the monitoring software) continually measures the radio signal strength values of the Gateway with respect to the departing router.

The trend recording of the communication signal strength reflects the degradation of the RSSI signals of the departing mobile router as it moves away from the gateway as shown in figure 5-30.

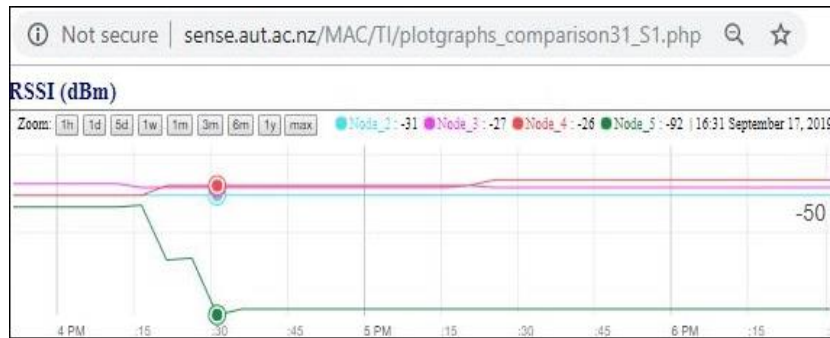


Figure 5-30 Data retrieved from SeNSe Lab's server depicting degradation of the RSSI signal of the departing router (node 5) as it moves away from the Gateway node.

In this particular case, virtualization of the process of detecting the event of departure of a mobile routing node moving away from the head router (or Gateway) (that could result in isolation of that particular part of the network) is focussed upon.

Both Table 5-4 and figure 5-31 incrementally reflect the synchronicity of the virtualization environment in (terms of) mimicking the dynamics occurring within the physical network in a similar way.

Table 5-4 Real RSSI values recorded for the departing router (with respect to the gateway) via physical experimentation.

Distance of the departing router with respect to the Gateway (m)	RSSI value of the departing router with respect to the Gateway (dBm)
1	-48
2	-57
3	-59
4	-61
5	-65
6	-66
7	-78 (Outside lab premises)
8	-92 (Outside lab premises)

Time	Mote	Message
10:49:46.382	ID:1	RSSI value of the departing router (with Node ID 4) with respect to the IoT-based gateway = -48 dBm
10:49:47.236	ID:2	RSSI value of the end device with node_ID=4 with respect to the IoT-based gateway = -48
10:50:42.382	ID:1	RSSI value of the departing router (with Node ID 4) with respect to the IoT-based gateway = -57 dBm
10:50:43.236	ID:2	RSSI value of the end device with node_ID=4 with respect to the IoT-based gateway = -57
10:51:38.382	ID:1	RSSI value of the departing router (with Node ID 4) with respect to the IoT-based gateway = -59 dBm
10:51:39.236	ID:2	RSSI value of the end device with node_ID=4 with respect to the IoT-based gateway = -59
10:52:34.382	ID:1	RSSI value of the departing router (with Node ID 4) with respect to the IoT-based gateway = -61 dBm
10:52:35.236	ID:2	RSSI value of the end device with node_ID=4 with respect to the IoT-based gateway = -61
10:53:30.382	ID:1	RSSI value of the departing router (with Node ID 4) with respect to the IoT-based gateway = -65 dBm
10:53:31.236	ID:2	RSSI value of the end device with node_ID=4 with respect to the IoT-based gateway = -65
10:54:26.382	ID:1	RSSI value of the departing router (with Node ID 4) with respect to the IoT-based gateway = -66 dBm
10:54:27.236	ID:2	RSSI value of the end device with node_ID=4 with respect to the IoT-based gateway = -66
10:55:22.382	ID:1	RSSI value of the departing router (with Node ID 4) with respect to the IoT-based gateway = -78 dBm
10:55:23.236	ID:2	RSSI value of the end device with node_ID=4 with respect to the IoT-based gateway = -78

Figure 5-31 Cooja nodes configured within the Cooja simulator with the real values obtained from physical experimentation.

The environment offered by Cooja to allow for the virtualization (for the router departure process in this case) is conducive for software-defined re-orchestration (at the virtual level) owing to the Contiki-provisioned configurability of virtual nodes using the modular functional components present within the ‘Data and Knowledge’ repository (as worked out by the pertinent knowledge components).

As alluded to earlier, dynamic software-defined re-orchestration to turn the elected leaf node to router within the virtualization environment involves Contiki-based software control to switching the functionality embedded within the successfully elected leaf node to that of a router node by means of implementing or activating the pertinent functional module within the code. The Contiki codes for the nodes have been included in the Appendix (within section A.1).

The sensed data (emanating from the physical network that is) stored within the Data repository within the server can not only be utilized for ‘Event Identification and Data Analysis’ purposes (for unusual events such as the aforementioned router departure event) but also for further examination and instantaneous influencing of the virtual network itself.

In this particular case, when the monitoring knowledge component detects that a pre-defined threshold RSSI value has been reached between the departing router and Gateway, it raises a trigger to initiate the process of electing a new router.

The number of communication transactions ensuing during the ‘Re-orchestration execution phase’ is the same as that of the virtual implementation. This example considers a case of rupture arising within a cloud-monitored sensor network owing to the gradual departure of the mobile router node away from the gateway node. This scenario necessitates one of the constituent leaf nodes among others that is most suitable to replace the departing router. This undergoes software-defined re-orchestration and assumes the role of a router to maintain network connectivity. The example clarifies a dynamic situation within the nodes’ mobility whereas network rupture may take place. Detection of such occurrence prompts the need for network re-orchestration. Available knowledge on such process at the cloud level offers the virtual service of recording the behaviour of such process and helps in monitoring and planning for required action on the physical network. While this offers a simplistic illustration in

facilitating the virtualization for given process (the physical organization and related dynamics), it reflects the possibility of developing virtual counterparts for various processes either at the early stages of the design or later and as further learning is introduced. On the other hand, the virtual network may represent a process, or it might cover complex case of various processes reflecting both the physical network of components (or nodes) and their dynamic interconnectivity. It has also put forth our notion of defining the process of demand for network re-orchestration into three different phases viz., ‘Data Analysis and Event-Identification’, ‘Re-orchestration-Planning’ and ‘Re-orchestration-Execution’.

Such segregation is directed towards our endeavour to clearly define the proposed organization’s capacity to detect network fragmentation in a somewhat pre-emptive fashion, and ‘planning’ for the necessary re-orchestrations in the background without disrupting the physical data collection process beforehand by means of relying on the cloud-based virtual and software resources. Furthermore, it also aids towards identification of the actual (physical) network downtime while implementing the last phase of Re-orchestration Execution.

5.8 Conclusion

The role played by Contiki-based software control in re-orchestrating both node and network level operations at the virtual environment provisioned through Contiki’s own virtual resource of Cooja simulator has been put forth through a number of incremental example implementations. By means of an example case of network fragmentation (caused by departure of a mobile router node away from the Gateway node implemented within Cooja), efforts towards studying the aspect of downtime incurred by the network whilst undergoing re-orchestration were pursued. Prior to this, classification of the various phases involved in the process of cloud-governed re-orchestration was ensued upon (using this router replacement case as an example). The entire re-orchestration process was broadly classified into three phases namely the ‘Data Analysis and Event-Identification’, ‘Re-orchestration-Planning’ and ‘Re-orchestration-Execution’ phases.

Besides, this particular example was also used to put forth certain efforts undertaken towards incorporating the element of reality into virtualization (by means of feeding real world data obtained from physical nodes into the corresponding virtual nodes) as an attempt to boost the accuracy of the soft re-orchestrations. Finally, the impact of varying three significant parameters on the overall re-orchestration latency was analysed by means of conduction of simulation-based experiments within Cooja.

Chapter 6

Use Case: Dynamic Re-orchestration of WSN Deployed for Forest Fire Monitoring Purposes

6.1 Introduction

WSNs deployed for forest fire monitoring purposes ought to cope with the dynamic network service and operational requirements that may emanate as a result of sudden fire outbreaks and potential network fragmentation events. The proposed ideology of a software defined wireless sensor network (SDWSN), backed by network virtualization offers good potential towards meeting the dynamic re-orchestration demands of such networks. This chapter attempts to incrementally demonstrate the above by means of considering an example case of network fragmentation (involving node-death of a router node caused by dead batteries) conducted on a preliminarily working virtual model.

Results indicate consistency of the cluster re-organizational outcomes, albeit accompanied by significant packet losses (for both up- scaled and downscaled network scenarios). While this may be an early stage of laying the ground for this attempt, there exists a significant potential for this direction of research as it links the two emerging technologies in identifying a potential solution that could contribute to the autonomous WSN-based forest fire monitoring solutions.

6.2 Demand for Dynamic Re-orchestration: Setting the Scene for the Example Case of Network Fragmentation in Consideration

A variety of factors tend to influence WSN deployments meant for forest fire monitoring purposes (both from the perspective of spatial distribution of nodes across the monitored region as well as the overall topological orientation of the same). These range from efficient energy consumption, distance amongst neighbouring nodes, prediction, rapid detection and location estimation of a fire outbreak, channel access or contention method, etc. [136].

Certain implementations tend to adopt the peer-to-peer topological approach [140, 148] whereas certain others have opted for a tree-based topological distribution of sensor nodes across the monitored region [137].

A schematic representing a cloud-based forest fire monitoring WSN deployment is as depicted within figure 6-1.

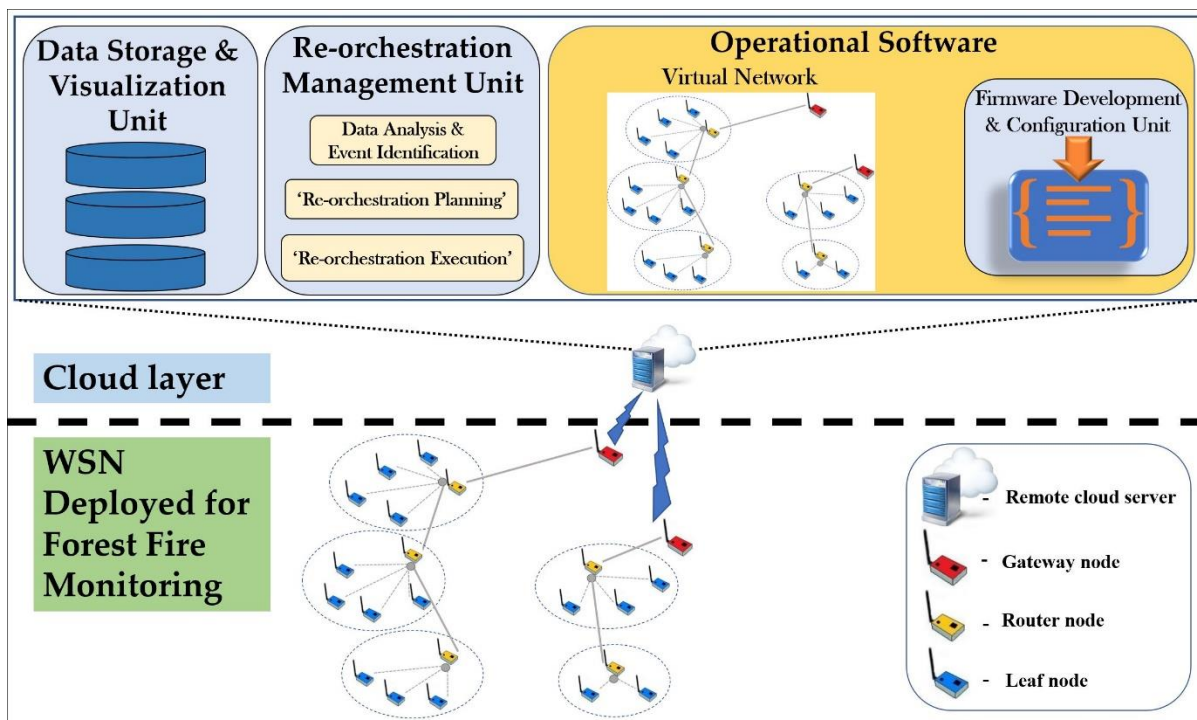


Figure 6-1 A schematic representing a cloud-based WSN organization for forest fire monitoring purposes.

Dynamics associated with forest fire monitoring may tend to involve of events of network fragmentation arising out of either nodes suffering damage as a result of fire or natural hazards or as a result of dying batteries from time-to-time. Sudden death of a router node acting as a cluster head of its particular group would most probably result in immediate and abrupt disruption of the cluster's outgoing sensed dataflow.

Certain dependent children members nodes, (of the dead or dying router node) might possibly, get disconnected from the group and thus, render that portion of the region unmonitored. Furthermore, such an event may also adversely impact the flow of sensed data emanating from the lower-level clusters present within the connectivity chain and vice versa (Refer figure 6-1).

Software-defined Wireless Sensor Networks (SDWSNs), in conjunction with virtualization could be conceived to be a highly prospective solution towards satisfying such transitory nature of re-orchestration demands in a dynamic manner. As emphasized throughout the course of this research work, cloud-enabled virtualization could play a significant role towards offering improved and flexible management of such large-scale deployments, whilst lending itself for data analysis, remote monitoring as well as prediction (of potential network fragmentation) purposes[149, 150].

It is deemed worthwhile to reiterate the assumption that nodes considered here are (pre-configured) with the requisite functional modules, enabling them to undertake or 'assume' any of the defined functional roles in a dynamic manner i.e., a leaf node may assume the router functional role (and cease to act as a leaf node upon receiving the requisite external signal or command).

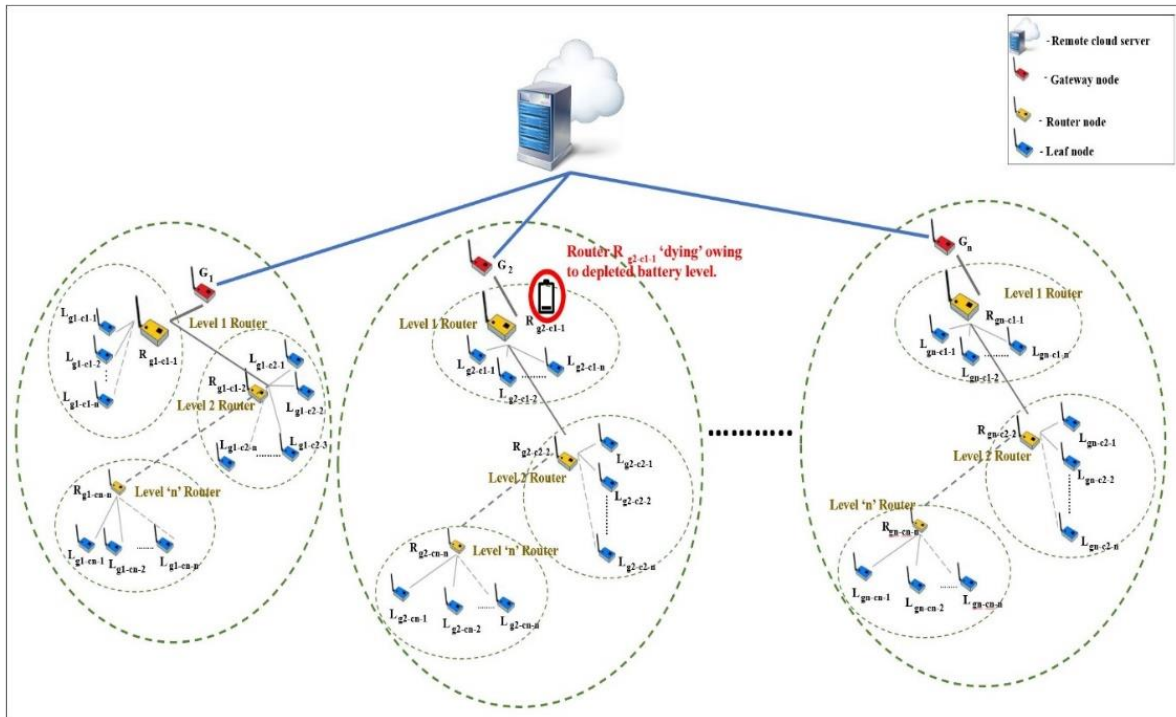


Figure 6-2 Example schematic of WSN deployed for forest fire monitoring purposes wherein a certain primary level router is suffering from low battery voltage and is about to die (requiring the network to undergo re-orchestration).

In keeping with this context, we direct our focus towards a typical use-case example of a router node, ‘acting’ as a cluster-head for its group of member leaf nodes, dying owing to depleted battery levels (Refer figure 6-1). This would tend to render its ‘dependent’ children (i.e., leaf) nodes unconnected. Hence, a new router node ought to be elected as its replacement. This replacement router node should also be able to accommodate any of the unconnected nodes. Network re-orchestration to recover from this situation could involve the virtual network residing within the cloud and elect a ‘potential replacement’ (i.e., the fittest possible leaf node), which will then take over the routing role. Electing a replacement router depends on the necessary measurements that are conducted by the relevant nodes and communicated through a sequence of messages among the main nodes in the network viz., the dying router, Gateway node, leaf nodes within the reach of the gateway node, leaf nodes beyond the reach of the gateway node and lastly, the routers of other clusters. A router fitness model for each such eligible candidate node ought to be formulated [151].

6.3 Formulation of Fitness Model

6.3.1 Description of Election Parameters

In order to clearly discern how the parameters would come into play, an attempt has been made to classify them as per two different stages (i.e., ‘pre-election’ and ‘during election’ stages).

1) Pre-election parameters

a) (Threshold) Battery level of (the dying Router) node

Information pertaining to depleted levels of battery power of any of the router nodes (by means of setting a threshold, below which any battery voltage value reported by it could trigger an alert) could be instrumental in offering an estimation as to which router node may stop functioning. Such information introduces an option of delay tolerance and thus, aids improvement of QoS.

Also, this data will offer information pertaining to the particular region monitored by the cluster of nodes of the dying router that may get disconnected and battery voltage level above the critical level.

b) Rate of battery depletion of (the dying Router) node

This data, in conjunction with the battery level data could be used to compute the time available before the node ceases to function. Alternatively, this piece of information could be used to compute the critical battery level value at which information pertaining to initiating the election be broadcasted.

c) A repository of addresses of nodes (encompassed by the dying router node)

The dying router has an updated repository or list of its constituent member leaf nodes that would be rendered disconnected from the rest of the network (upon its death). These nodes, if falling within the communication range of the gateway node, ought to be informed by it to partake within the process of electing a replacement router node from amongst them. If unsuccessful, they are to search for a new parent.

Within the framework of an ‘aware’ network, information derived from these parameters could be useful in terms of acquiring pre-knowledge of

- the eventuality of a particular parent router about to die [so as to duly initiate (pre-emptive) router election process].
- the estimated time at which router will cease all communication operations

Utilization of such prior knowledge to activate the election process will either prevent loss of network connectivity or mitigate the time for which the affected leaf nodes remain unconnected.

2) Election-based (Decisive) parameters

a) Radio Signal Strength of a potential router node from the gateway node

Herein, radio signal strength values of each of the participant nodes (with respect to the gateway) are taken into account (eliminating the need to take into consideration the calculation of respective distances from it). All participant leaf nodes communicate their respective RSSI values to the gateway node reflecting a more accurate estimate of their connectivity with the gateway node [151].

b) Radio Signal Strength between the potential replacement router node and the constituent leaf nodes of the cluster that may be deprived of network connectivity

The router eligible to partake within the election process, along with the leaf nodes falling within the communication range of the gateway node as well as routers acting as cluster head

for other clusters of leaf nodes as well as eligible routers in the election process, attempt to establish connection with leaf nodes (that may be rendered disconnected as a result of the dying cluster head node) that fall outside the communication range of the gateway. This aids identification of such unreachable leaf nodes would have to be accommodated by a nearby router-cluster-head node, ensuring its connectivity within the overall network.

c) Ability of potential replacement router to act as a cluster-head for maximum number of leaf nodes that may be rendered disconnected from the network

This parameter has been considered herein on the premise that the replacement router ought to cater to able to act as a cluster-head for as many leaf nodes that may drop out of the connectivity chain (as a result of the ensuing ‘node-death’-based fragmentation event) as possible. This may also divulge information pertaining to the number of children that may establish connections with the prospective replacement router. Any participant leaf node with RSSI value below a certain reliable threshold level would refrain from forming a connection with a given participant router node.

d) Number of existing children nodes already connected to the potential router node

Each router node has a certain maximum capacity of connected children leaf nodes. For example, a maximum of 3 nodes may be specified. Lesser the number of children connected to a potential router, greater would be the possibility of it winning the election process.

e) Level of battery power of the potential replacement router node

Prevailing levels of battery power of the participant nodes have also been taken into account here. This factor ought to be given due weightage along with the aforementioned parameters owing to its significance towards ensuring reliable connectivity over the long-term.

Figure 6-2 shows the screenshot of the virtual network showing only the region around the cluster affected by the dying router node. Here, the affected leaf nodes participate in the election process. If successful, they will assume the functional role of a router node (upon being remotely invoked via requisite external commands, for the same).

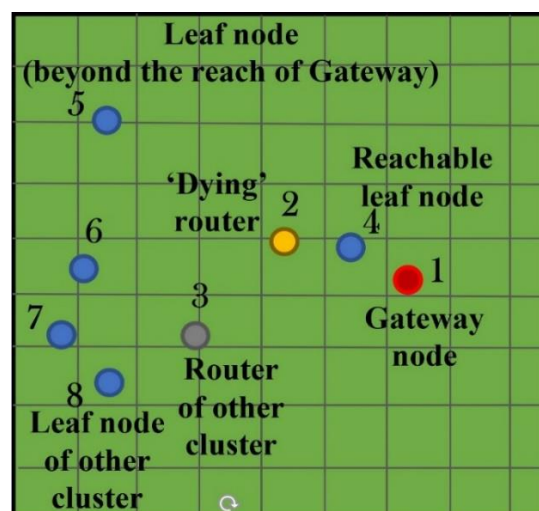


Figure 6-3 Screenshot of the virtual network showing only the area of the overall WSN consisting of the cluster wherein member leaf nodes are affected by the dying router node.

6.3.2 Sequence of Messages for Electing a New Router

The process of election of a replacement router node commences once the alert triggered by the router node (owing to its battery value falling below a certain set threshold value) is disseminated amongst the various participant nodes [151]. Thereafter, each of the participant nodes ensue upon communicating messages (based on the aforementioned parameters) to the gateway node in order for it (the gateway node) to compute the (normalized) fitness value for each of them. These values are subsequently compared to determine the node with the highest fitness value which is then declared to be the replacement router node. Descriptions of the sequence of messages exchanged amongst the various nodes during the ‘election’ process is provided below.

- 1) Once the battery level of certain router node falls below the set threshold value, it calculates its remaining lifetime of the dying router node and transmits this piece of information along with the addresses of its children nodes (that will be disconnected from the rest of the network upon its death) to the gateway, the (‘reachable’ as well as ‘unreachable’) leaf nodes, as well as the routers of other clusters, through message ‘M_{RD}’ as depicted in figure 6-3 [151].
- 2) Once the gateway node receives this message from the dying router, it attempts to ‘broadcasts’ message ‘M_{G-L}’ to all of the leaf member nodes belonging to the cluster yet governed by the dying router node. This particular message is aimed at evoking acknowledgement messages consisting of respective radio signal strength values (denoted by ‘M_{RC-L}’) from all the reachable leaf nodes, enabling it to identify and segregate them from the unreachable ones. Once acknowledgment messages from all such reachable nodes are received, the gateway node responds to them via message ‘M_{G-RL}’, essentially informing them regarding their eligibility as participant nodes within the election process.
- 3) In an attempt to check their radio signal strength values with respect to the leaf nodes of the dying router, the ‘routers of other clusters’, transmit radio messages denoted by ‘M_{ROC-RL}’ to them. Upon receiving ‘acknowledgment’ message(s), denoted by ‘M_{RL-ROC}’, from them (provided they fall within their respective communication ranges), these ‘routers of other clusters’ transmit message ‘M_{ROC-G}’ to the gateway node updating it with the number of leaf nodes within their communication range, and the average value of their radio signal strength values with each of the leaf nodes affected by the death of their governing router.
- 4) The participant leaf nodes (capable of assuming the router function and belonging to the cluster of the ‘dying’ node) temporarily switch over to the role a router node to broadcast a signal denoted by ‘M_{RCL-L}’ amongst each other. The intention here is to determine how many of the participant leaf nodes fall within the communication range of each of them. This information pertaining to the number of possible connections (obtained when acknowledgment message(s) ‘M_{L-RCL}’ in response to message ‘M_{RCL-L}’ are received by each of the ‘affected leaf nodes’), is ‘communicated’ to the ‘gateway’ node via transmission of the message ‘M_{RL-G}’.
- 5) Based on the information gathered from all the ‘participant’ (reachable ‘leaf’ nodes as well as ‘routers ‘of other clusters), the gateway node computes the fitness value for each of them. The participant node with the ‘highest fitness value’ is elected as the new replacement router, the node ID of which is subsequently broadcasted to all the participant nodes (via messages ‘M_{G-RL}’ and ‘M_{G-R}’). Information regarding the leaf nodes that will be governed by the newly elected router is also included within these messages.

The process involves two models. One is directed towards election of a leaf node that is 'reachable' by the gateway node and can accommodate as many leaf nodes as possible that are directly affected by the death of their governing router. The second vies to find a possible other router node that could accommodate for the remaining unconnected nodes.

6) Once the connection between the newly elected replacement router and its leaf member nodes is established (denoted by message 'M_{ERCL-L}') and service dataflow within the re-orchestrated cluster resumes, the routers of other clusters direct the relevant leaf nodes to do the same (i.e., resume normal flow of data) in accordance with the re-orchestrated network arrangement.

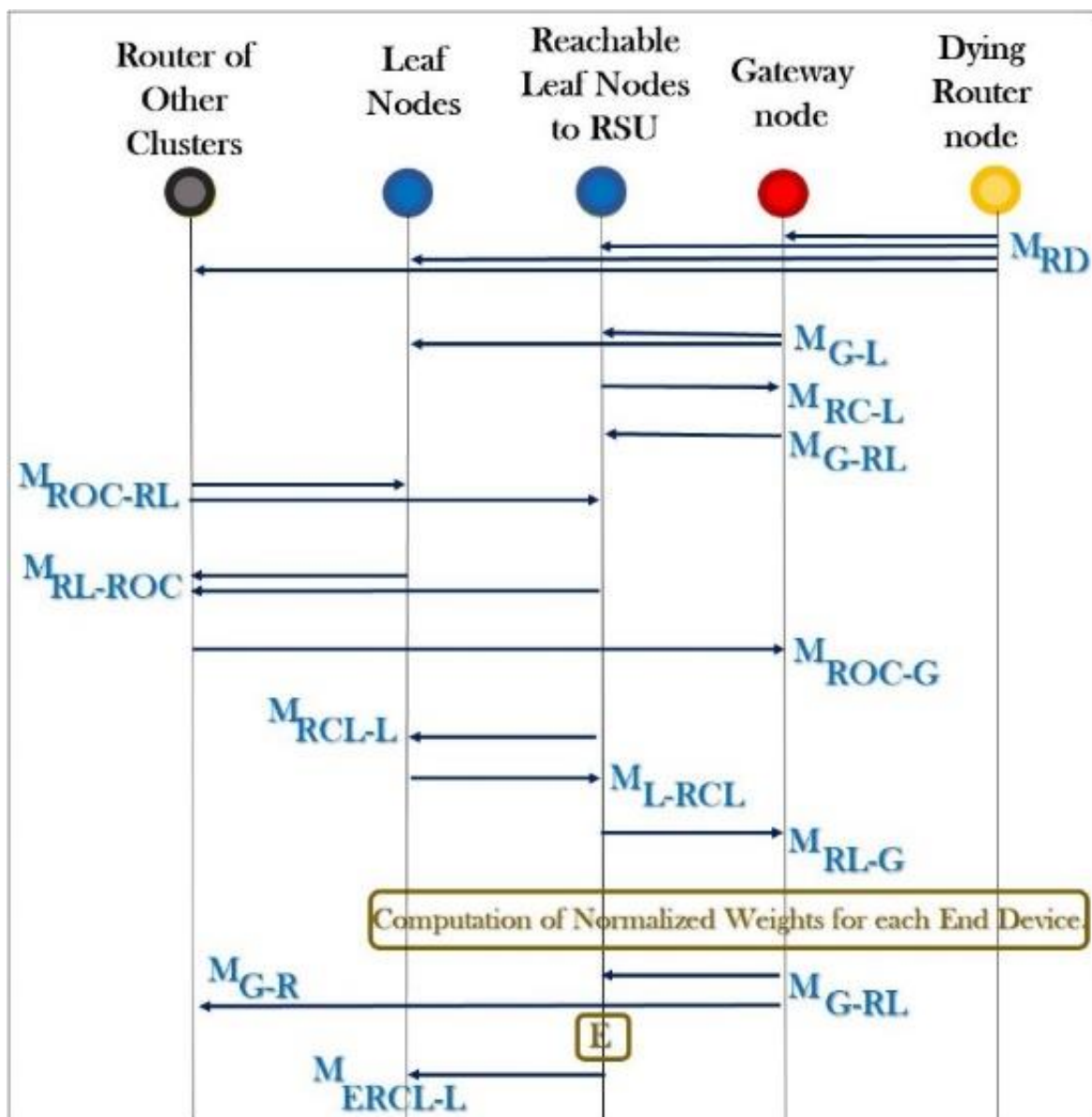


Figure 6-4 Sequence diagram depicting the various messages exchanged amongst the various nodes partaking in the election process.

6.3.3 Model Formulated Towards Determining the Fitness Value of Participant Nodes

The model for the selection process has two main sub-processes associated with it. These should result with the identification of the necessary router(s) that establish connectivity with the leaf nodes of the departing router. The first sub-process attempts to identify a router node among these leaf nodes. Emphasis has been laid on leaf nodes falling within the communication range of the ‘gateway’ (referred to as reachable nodes) that will compete amongst one another for the ‘replacement router’ node. The process may end up with the selection of a node that facilitates routing to a number of relevant leaf nodes. This might be any number of nodes between the maximum (i.e., all nodes) and zero (i.e., no node has been selected).

Based on the parameters considered within section 6.3.2, the model formulated towards determining the fitness value of a given participant node is as follows:

$$W_f = \{[NW_j \times RSSI_{RCL-G}] + [NW_{(j+1)} \times RSSI_{RCL-L}] + [NW_{(j+2)} \times C_{CL-RCL}] + [NW_{(j+3)} \times B_{RCL}]\} \quad [151]$$

where, $NW_j = W_j \times f_j$,

‘ f_j ’ denotes the ‘fiddle’ factor associated with a (respective) particular variable,

$RSSI_{RCL-G}$ denotes the radio signal strength of a given participant reachable leaf node with respect to the gateway node,

$RSSI_{RCL-L}$ denotes the radio signal strength of a given participant reachable leaf node with respect to the other leaf member nodes,

C_{CL-RCL} denotes the quantity of member leaf nodes that fall within the communication of the reachable leaf node and

B_{RCL} represents the battery level of the reachable member leaf node (i.e., RCL).

Each of the above ‘fiddle’ factors associated with their respective parameters have been considered independently of each other as a means to normalize the weightage allocated to each of them, and thereby the resultant fitness value, i.e., ‘ W_f ’ [151]. (As a means to elaborate on the above formulated fitness model, a sample router fitness calculator table for a given participant node towards electing replacement router (to replace the dying router node) has been provided in section A.6 of the appendix.) However, even though the weight for each impact factor is chosen as unity to start with, it is liable to be tampered with as part of the ongoing learning process (which is part of the cloud background operation on the virtual network).

The second sub-process is relevant to the possible distribution of the remaining unallocated leaf nodes. The criterion here considers balancing the load through using the number of existing leaf nodes connected to each router and is expressed as below:

$$W_v = \{[NW_j \times RSSI_{ROC-RSU}] + [NW_{(j+1)} \times RSSI_{ROC-L}] + [NW_{(j+2)} \times C_{MCL-ROC}] + [NW_{(j+3)} \times B_{ROC}]\}$$

where, $RSSI_{ROC-RSU}$ represents the received signal strength indication for ROC (router of other cluster) with respect to the gateway node,

$RSSI_{ROC-L}$ represents the received signal strength indication,

for ROC with respect to other leaf nodes,

$C_{MCL-ROC}$ represents the maximum number of leaf nodes that are reached by ROC node(s) considering the existing connected leaf nodes and

B_{ROC} represents the battery level of ROC node(s).

6.3.4 Implementation on Virtual Platform and Results

As a means to confirm the workability of the conceptual model so formulated, the concept has been incrementally tested on a simulated stationary scenario using the Cooja simulator. Consider figure 6-4 [151], which depicts the implementation and modeling of the aforementioned simplistic, notional forest fire monitoring sensor network-based clustering scenario using Contiki-Cooja simulator.

Herein, the dying router is disseminating the 'M_{RD}' message to all the participant nodes. Through the knowledge of the battery level, rate of battery depletion, the time could be predicted, thus helping in defining the time available for electing a replacement router node.

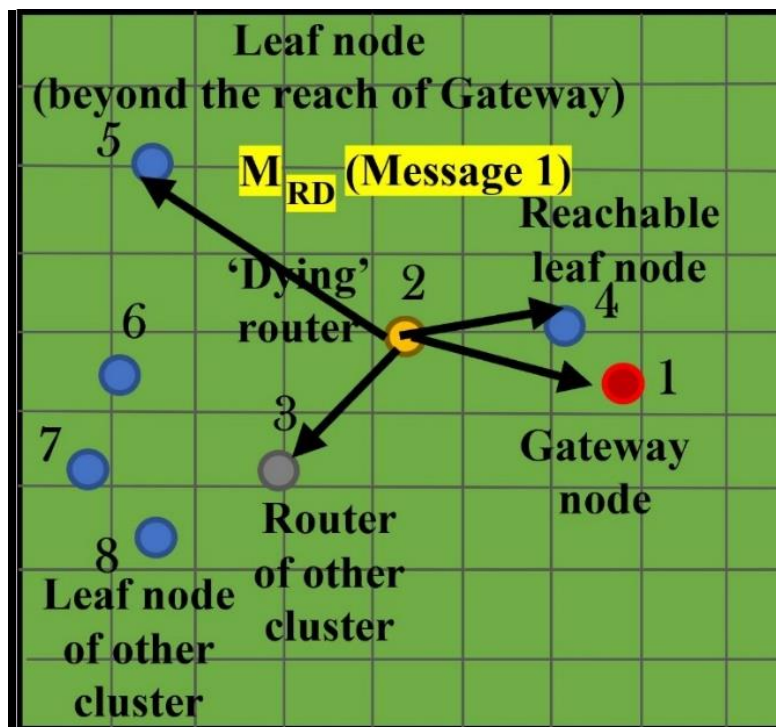


Figure 6-5 Virtual network implementation wherein the dying router broadcasts message 'M_{RD}' to all the relevant nodes.

At this preliminary testing stage, the consistency of the model, (with respect to the correct delivery of outcome of the election process) was observed for the following case which was deliberately considered and repeated for 100 times so as to check if there is any irregularity in the election result (i.e., whether the same node gets elected with the same fitness model each time. This would confirm the veracity of the working model). Another reason for repeating the tests these many times was to check for any packet losses as a result of the communication process taking paces amongst the nodes.

This early stage of exploring the model also involved scaling up i.e., varying the number of participant reachable leaf nodes from one to six nodes as shown in figure 6-5 [151], as an attempt to observe the capacity of the model to deal with a greater number of nodes.

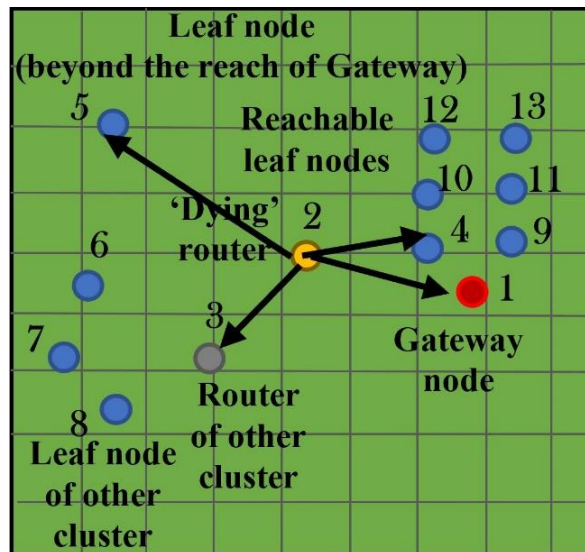


Figure 6-6 Static model implemented within the virtual Network: Number of participant reachable leaf child nodes scaled up to six nodes.

While re-running the simulation tests for the static case for 100 times revealed no irregularity or ‘variability’ with respect to the ‘election’ outcome (see figure 6-6), the process accompanied a sizeable amount of packet losses. The cause for such significant and unanticipated packet losses is being investigated.

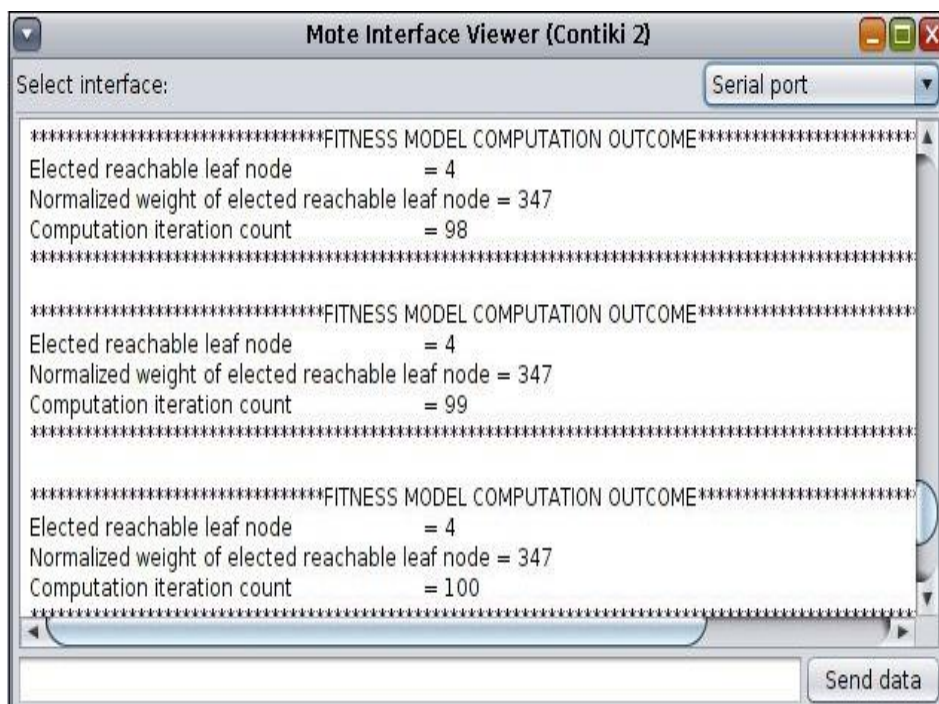


Figure 6-7 Cooja simulator outcome confirming consistence of the result.

6.4 Conclusion

The chapter explores the organization of wireless software defined sensor networks with emphasis on self-organization in the event of anticipated partial disruption to a WSN deployed for forest fire monitoring purposes. The cloud offers efficient operational environment for re-orchestration. Owing to the resources of virtualization, stored historical data and learning tools and methods, the cloud could then be made to explore the scope for future improvement in the operation with different software scenarios. While putting forth a case for dynamically changing network, a network meant for forest fire monitoring purposes is made the subject of focus in resolving the fragmentation occurring as a result of the death of a key connectivity component within the network (in this case, a router node). Incremental testing has been made successfully here. However, scope exists towards conduction of more thorough and integrated tests whilst exploring the possibility of reflecting the role of the cloud as a look-ahead component for the organization of dynamic improvements and playing the role of dynamic re-orchestration.

Chapter 7

Conclusion

7.1 Introduction

As a means to cater for the crucial characteristic requisite of operational flexibility for wireless sensor networks (WSN), an architectural solution leveraging upon cloud-based virtualization and software resources has been proposed. Herein, a cyber-physical based sensor network organization has been conceptualized that allows for interactive collaboration between the cloud (equipped with the necessary data storage, virtualization, re-orchestration management and firmware development and configuration resources) and physical layers towards facilitating for the necessary identification, planning and execution of the re-orchestration process as necessitated by prevailing service requirements. This has been deemed as a prospective solution for the realization of software-defined sensor network systems. Such a cloud-supported framework has opened new vistas towards remote monitoring, hardware-independent soft-trialing, besides augmented flexible control and management, i.e., dynamic re-orchestration of the underlying physical network.

This thesis pursues towards collaborative interaction amongst the aforesaid technological resources hosted by the cloud and the physical layer under the aegis of a cyber physical (architectural) framework, wherein well-defined, modular constituent functional components could be virtualized and subjected to multifarious software-defined re-orchestration. Embodiment of the requisite potential and intelligence to progressively evolve with experience and flexibly engage with the monitored external phenomena and/or meet the dynamic service requirements is also an important consideration within the architecture so proposed.

The thesis takes into cognizance the efficacy of subjecting the abstractions, i.e., virtualized forms of the three core functionalities (i.e., the gateway' function, router' function and the leaf node function, along with their associated sub-functionalities as well as certain additional WSN-allied functions), stored as software 'functional' modules (within the cloud layer) to software-defined re-orchestrations under the aegis of the virtualization environment. That adoption of the (relatively simplistic yet fairly advantageous) re-configurability approach towards realization of flexible node and network-level re-orchestrations (at both the physical and virtual levels) is a somewhat more viable approach when compared with re-programmability-based approaches (such as OTAP) is brought to the fore. The aspect of augmenting the flexibility of a (resource-rich) node via creation and integration of a library of reusable firmware modules within its firmware (wherein requisite new reusable software modules could be added from time to time) is discussed within the concept chapter. Advancements made within embedded technology have to a certain extent eliminated the impracticability associated with direct implementation of the softwarization paradigm to low power wireless sensor-cum-transceiver devices.

By means of leveraging upon its abundant resources encompassed by it viz., virtualization, software control, 'Data and Knowledge Repository', etc.), the cloud can figure out the sequence of messages to be executed to re-orchestrate the physical WSN network whilst incurring

reduced downtime and ensure that the node meant to undergo re-orchestration is capable of accommodating for additional functions.

The proposed research aims to significantly contribute to the existing state-of-the-art in terms of converging upon a modular, cloud-supported software defined sensor network organization that can flexibly re-orchestrate its behaviour at both the node and network-operational levels (by virtue of the re-configurability approach) at the virtual level first (prior to implementation at the physical layer) whilst adhering to the three-phased re-orchestration strategy (outlined in chapter 4).

7.2 Conclusion

Converging upon the objective of a flexible and adaptive WSN system entailed dealing with physical WSN setup on one hand and cloud-level virtualization cum cognitive analytics on the other. In a bid to create such a flexible WSN organization endowed with virtualization capability, it is necessary to secure an ‘accurate’ and ‘precise’ logical correlation between the physical and virtual WSN environments. The Contiki-based Cooja fulfils this fundamental requirement via using the same Contiki-generated firmware for ‘compiling’, configuring and creating the virtual Cooja motes as that used for configuring the physical ‘TI CC2538’ hardware motes.

The cloud-based virtualization unit, whilst operating in conjunction with the ‘Data and Knowledge repository’, offers itself as an avenue for testing the soft-trials of such flexible ‘WSN function’ reformulations so as to converge upon the most conducive WSN re-orchestrations. Such collaborative operation amongst the two cloud components tends to be portentous towards the overall flexible operation of the network in a dynamic fashion.

The examples depicted within this thesis reflect the saliency of Contiki-Cooja facilitated virtualization environment as a means to mimic the dynamics of the underlying physical sensor network, thereby providing for (continual) remote monitoring, planning or facilitating the necessary re-orchestrations (at the cloud level), (if required to align the sensor network dynamics with that of the external monitored phenomenon), exploring and exploiting the possible degrees of freedom (within the network architecture which could act on manipulating the network parameters and hence controlling the network operation, without disrupting its key operational process of data collection), etc. Besides these advantages, it is worthwhile to mention that Cooja provisioned virtualization environment serves to significantly not only optimize the flexible re-orchestration process of physical sensor network but also analyse the implications of the re-orchestration process on the network serviceability beforehand, i.e., the extent of downtime experienced by the sensor network.

The overall re-orchestration process can be said to entail a number of key phases which the system is required to undergo sequentially (until the network can be said to have been completely re-orchestrated). By means of an example case of sensor network re-orchestration implemented within the virtual network offered by Cooja, the three phases of re-orchestration viz., ‘Data Analysis and Event-Identification’, ‘Re-orchestration-Planning’ and ‘Re-orchestration-Execution’ have been discussed.

Bulk of the computational aspect (pertaining to the re-orchestrations to be applied) takes place in the cloud during the Re-orchestration Planning phase. Although the Re-orchestration process spans over three phases, the actual downtime is confined to the ‘Re-orchestration execution’ phase alone. The time for which a network suffers downtime tends to depend on factors such as ‘number of nodes’, number of hops, data communication rate, channel access method adopted, Pre-defined fixed time interval between two consecutive time slots (when ‘scheduled-based channel access method’ is adopted), number of messages exchanged amongst the nodes (during the re-orchestration process), size of data packets being transmitted, transmission time, switching time (i.e., time taken by a node to switch to different functional role) and latency entailed by a single hop.

The results pertaining to network downtime for the specific cases of network re-orchestration examples depicted in this thesis (extracted via implementation within the virtual Cooja environment) are relative. In order to obtain near-accurate results, it necessary to take into account the various aforementioned factors, especially the channel access method, data communication rate as well as routing protocol implemented, including real-world factors affecting the communication operations.

In order to reflect the applicability of the proposed approach or concept, the real-life use case of forest fire monitoring’ has been focused upon within this thesis. The caliber of the said framework (comprising of ‘Data and Knowledge repository’, consisting of the application-specific knowledge components, software modules, etc., working in tandem with the virtualization unit) in dynamically re-orchestrating network dynamics in the event of outbreak of forest fire is demonstrated. It is anticipated that such an approach could benefit a wide variety of WSN application domains (ranging from highly dynamic mobile sensor networks such as the vehicular or drone-based sensor network implementations to the largely static WSN deployments viz., environmental sensing, smart home applications, etc.) towards pre-emptive detection of an event necessitating re-orchestration and allowing for the same whilst reducing the downtime associated with it (re-orchestration process) to a considerable extent.

7.3 Future work

7.3.1 Realization and Incorporation of the Aspect of Digital Twin

In line with the current endeavour to converge upon a flexible software-defined sensor network organization with enhanced accuracy and (dynamic) responsiveness towards real-time re-orchestration demands, progression from virtualization to a ‘Network Digital Twin’ is deemed to be imperative. This forms an important and challenging part of the future work.

Realization of digital twin primarily involves connecting the physical sensors with their virtual counterparts. (Deemed to be more potent than mere virtualized entities,) network digital twins can not only enable forecasting of an estimated outcome of a particular soft-trial, i.e., virtual re-orchestration, but also provide status of the ongoing dynamics of the physical network in real time [71].

So far as our specific research interests are concerned, it is conceived that figuring out a way to ‘link’ the data emanated by the physical CC2538 sensors to their virtual Cooja counterpart nodes in real time could increase the level of accuracy (via incorporating the element of reality within) the virtualization feature provisioned by Cooja.

7.3.2 Re-programmability-based Approaches

As a possible alternative to re-configurability-based approaches, dynamic re-programmability-based approach involving dissemination of the requisite software modules onto the desired nodes (for the purposes of assembly or re-assembly of the same within that particular node) via OTAP (Over-The-Air-Programming)-based techniques viz., paging, delta updates and methods such as sprinkler, deluge, etc., could also be looked at and implemented. [147]. The potential offered by OTAP-based protocols for WSNs as a means to implement the desired re-orchestrations for multiple sensor nodes ought to be exploited. Contiki does offer support for OTAP, provided the network is a homogeneous (i.e., consisting of the same target hardware employed as nodes) one and running the same OS and firmware (i.e., Contiki) [152]. Incorporation of OTAP-based approaches eliminates the need to have in place a management software running on top of the nodes, besides the API needed for the same [153]. It is however important that due measures towards addressing the challenges associated with OTAP-based protocols (e.g., time taken for re-programming of a node, are taken prior to implementation onto real-world networks).

7.3.3 Edge-Computing Aspect

The research concept, in its current state, relies almost entirely on the cloud resources for the computational (or knowledge-crunching-based) aspects. However, the prospect of availing the benefits offered by ‘edge computing’ must be investigated.

Although all the knowledge components can be housed within the cloud, it is seemed rather logical to offload certain feasible knowledge components to the edge level devices, e.g., Gateway node(s).

It is envisaged that offloading certain knowledge components responsible for mild computations at the sub-cloud level could offer somewhat further reduce the number of communication transactions i.e., exchanges (amongst the various network components) and hence, the latency. This, in turn, could also significantly lessen the prospects of packet loss.

This incentivizes further investigation of the implications of allocation or migration of knowledge components to the sub-cloud levels of fog and edge (depending on the ability of the edge device e.g., gateway, to assume them) on overall network downtime.

References

- [1] Jamil, M. S., Jamil, M. A., Mazhar, A., Ikram, A., Ahmed, A. and Munawar, U. "Smart environment monitoring system by employing wireless sensor networks on vehicles for pollution free smart cities" in *Procedia Engineering*, pp. 480-484, 2015.
- [2] Pirbhulal, S., Zhang, H., E Alahi, M. E., Ghayvat, H., Mukhopadhyay, S. C., Zhang, Y. T. and Wu, W., "A novel secure IoT-based smart home automation system using a wireless sensor network," in *Sensors*, 2017.
- [3] Cao-Hoang, T. and Duy, C. N., "Environment monitoring system for agricultural application based on wireless sensor network," in *Seventh International Conference on Information Science and Technology (ICIST)* pp. 99-102. IEEE, 2017.
- [4] Xie, H., Yan, Z., Yao, Z. and Atiquzzaman, M., "Data collection for security measurement in wireless sensor networks: A survey" in *IEEE Internet of Things Journal*, 6(2), pp. 2205-2224, 2018.
- [5] Fu, X., Fortino, G., Pace, P., Aloï, G. and Li, W., "Environment-fusion multipath routing protocol for wireless sensor networks," in *Information Fusion*, 53, pp. 4-19. Telecommunication Symposium, 2020).
- [6] Wang, Q. and Balasingham, I., "Wireless sensor networks-an introduction" in *Wireless sensor networks: application-centric design*, pp. 1-14, 2010.
- [7] Kim, D. S. and Chung, Y. J. "Self-organization routing protocol supporting mobile nodes for wireless sensor network" in *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)*, Vol. 2, pp. 622-626, IEEE, 2006.
- [8] Ye, D., Gong, D. and Wang, W., "Application of wireless sensor networks in environmental monitoring," in *2nd international conference on power electronics and intelligent transportation system (PEITS)*, Vol. 1, pp. 205-208, IEEE, 2009.
- [9] Mittal, R. and Bhatia, M. S., "Wireless sensor networks for monitoring the environmental activities" in *2010 IEEE International Conference on Computational Intelligence and Computing Research*, pp. 1-5. IEEE, 2010.
- [10] Othman, M. F. and Shazali, K., "Wireless sensor network applications: A study in environment monitoring system," in *Procedia Engineering*, 41, pp. 1204-1210, 2012.
- [11] Somov, A., Shadrin, D., Fastovets, I., Nikitin, A., Matveev, S., Seledets, I. and Hrinchuk, O., "Pervasive Agriculture: IoT-Enabled Greenhouse for Plant Growth Control," in *IEEE Pervasive Computing*, vol. 17, no. 4, pp. 65-75, 2018.
- [12] Hassan, M. M., Albakr, H. S. and Al-Dossari, H., "A cloud-assisted Internet of things framework for pervasive healthcare in smart city environment." in *Proceedings of the 1st International Workshop on Emerging Multimedia Applications and Services for Smart Cities*, pp. 9-13, 2014.
- [13] Haggi, M., Neubert, S., Geissler, A., Fleischer, H., Stoll, N., Stoll, R. and Thurow, K., "A Flexible and Pervasive IoT-Based Healthcare Platform for Physiological and Environmental Parameters Monitoring," in *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5628-5647, 2020.
- [14] Porambage, P., Schmitt, C., Kumar, P., Gurtov, A. and Ylianttila, M. "PAAuthKey: A Pervasive Authentication Protocol and Key Establishment Scheme for Wireless Sensor Networks in Distributed IoT Applications," in *International Journal of Distributed Sensor Networks*, p. 14, 2014.

- [15] Tracey, D. and Sreenan, C., "A Holistic Architecture for the Internet of Things, Sensing Services and Big Data," in *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, Delft, Netherlands, 2013, pp. 546-553.
- [16] Zhu, Q., Wang, R., Chen, Q., Liu Y. and Qin, W., "IOT Gateway: Bridging Wireless Sensor Networks into Internet of Things," in *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Hong Kong, China, pp. 347-352, 2010.
- [17] Chen, H., Jia, X. and Li, H., "A brief introduction to IoT gateway," in *IET International Conference on Communication Technology and Application (ICCTA 2011)*, Beijing, pp. 610-613, 2011.
- [18] Zhong, C., Zhu, Z. and Huang, R., "Study on the IOT Architecture and Gateway Technology," in *14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, Guiyang, China, pp. 196-199, 2015.
- [19] Woo, M. W., Lee, J. and Park, K., "A reliable IoT system for personal healthcare devices", in *Future Generation Computer Systems*, 78, pp. 626-640, 2018.
- [20] Guoqiang, S., Yanming, C., Chao, Z. and Yanxu, Z., "Design and Implementation of a Smart IoT Gateway," in *IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, Beijing, China, pp. 720-723, 2013.
- [21] McGrath M.J. and Scanaill C.N., "Sensor Network Topologies and Design Considerations" in *Sensor Technologies*. Apress, Berkeley, CA, pp. 79-95, 2014.
- [22] Marais, J., Malekian, R., Ye, N. and Wang, R., "A Review of the Topologies Used in Smart Water Meter Networks: A Wireless Sensor Network Application," in *Journal of Sensors*, 2016.
- [23] Shi, B., Sreeram, V., Zhao, D., Duan, S. and Jiang, J., "A wireless sensor network-based monitoring system for freshwater fishpond aquaculture" in *Biosystems engineering*, Vol. 172, pp. 57-66, 2018.
- [24] Ammar, M.B. and Souissi, R., "A New Approach based on Wireless Sensor Network and Fuzzy Logic for Forest Fire Detection" in *International Journal of Computer Applications*, Vol. 89, pp. 48-55, 2014.
- [25] Acharyya, I. S., Al-Anbuky, A. and Sivaramakrishnan, S., "Software-Defined Sensor Networks: Towards Flexible Architecture Supported by Virtualization," in *Global IoT Summit (GIoTS)*, Aarhus, Denmark, pp. 1-4, 2019.
- [26] Acharyya, I. S., and Al-Anbuky, A., "Software-defined Wireless Sensor Network: WSN Virtualization and Network Re-orchestration" in *SMARTGREENS*, pp. 79-90, 2020.
- [27] Ali, A., Ming, Y., Chakraborty, S. and Iram, S., "A comprehensive survey on real-time applications of WSN" in *Future Internet*, Vol. 9, Issue 4, 2017.
- [28] Minaie, A., Sanati-Mehrziy, A., Sanati-Mehrziy, P. and Sanati-Mehrziy, R., "Application of wireless sensor networks in health care system" in *age*, 23(1), pp. 1-12, 2013.
- [29] Grover, K., Kahali, D., Verma, S. and Subramanian, B., "WSN-based system for forest fire detection and mitigation," in *Emerging Technologies for Agriculture and Environment*, pp. 249-260, Springer, Singapore, 2020.
- [30] Kadir, E. A., Rosa, S. L. and Yulianti, A., "Application of WSNs for detection land and forest fire in Riau Province Indonesia," in *2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*, pp. 25-28, IEEE, 2018.
- [31] Trivedi, K. and Srivastava, A. K., "An energy efficient framework for detection and monitoring of forest fire using mobile agent in wireless sensor networks," in *2014 IEEE*

- International Conference on Computational Intelligence and Computing Research*, pp. 1-4, IEEE, 2014.
- [32] Chi, Q., Yan, H., Zhang, C., Pang, Z. and Da Xu, L., "A reconfigurable smart sensor interface for industrial WSN in IoT environment", in *IEEE transactions on industrial informatics*, 10(2), pp. 1417-1425, 2014.
- [33] Duan, Y., Li, W., Fu, X., Luo, Y. and Yang, L., "A methodology for reliability of WSN based on software defined network in adaptive industrial environment." in *IEEE/CAA Journal of Automatica Sinica*, 5(1), pp. 74-82, 2017.
- [34] Begum, K. and Dixit, S., "Industrial WSN using IoT: A survey" in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pp. 499-504, IEEE, 2016.
- [35] Lu, X., Wang, S., Li, W., Jiang, P. and Zhang, C., "Development of a WSN based real time energy monitoring platform for industrial applications" in *2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 337-342, IEEE, 2015.
- [36] Lazarescu, M. T., "Design of a WSN Platform for Long-Term Environmental Monitoring for IoT Applications," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 3, no. 1, pp. 45-54, 2013.
- [37] Leligou, H. C., Redondo, L. Zahariadis, T., Retamosa, D. R., Karkazis, P., Papaefstathiou, I. and Voliotis, S., "Reconfiguration in Wireless Sensor Networks," in *2010 Developments in E-systems Engineering*, London, UK, pp. 59-63, 2010.
- [38] Satija, S., Sharma, T. and Bhushan, B., "Innovative approach to Wireless Sensor Networks: SD-WSN," in *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Greater Noida, India, pp. 170-175, 2019.
- [39] Ndiaye M., Hancke G.P. and Abu-Mahfouz A.M., "Software Defined Networking for Improved Wireless Sensor Network Management: A Survey" in *Sensors*, Vol.17, Issue 5, 2017.
- [40] Violettas, G.,Theodorou, T., Petridou, S., Tsioukas, A. and Mamatas, L., "Demo Abstract: An Experimentation Facility Enabling Flexible Network Control for the Internet of Things," in *IEEE Conference on Computer Communications Workshops*, Atlanta, GA, pp. 992-993. 2017.
- [41] Acharyya, I. and Al-Anbuky, A., "Towards wireless sensor network softwarization," in *IEEE NetSoft Conference and Workshops NetSoft 2016*, Seoul, Korea Republic, pp. 378-383, 2016.
- [42] Ezdiani, S., Acharyya, I., Sivakumar S. and Al-Anbuky, A., "Wireless Sensor Network Softwarization: Towards WSN Adaptive QoS" in *IEEE Internet of Things Journal* 2017, Vol. 4, No. 5, pp. 1517-1527, 2017.
- [43] Eronu, E., Misra, S. and Aibinu, M., "Reconfiguration approaches in Wireless Sensor Network: Issues and challenges," in *IEEE International Conference on Emerging & Sustainable Technologies for Power & ICT in a Developing Society NIGERCON*, Owerri, 2013.
- [44] Huang, M. and Yu, B., "Towards General Software-Defined Wireless Sensor Networks" in *Proceedings of the 4th IEEE International Conference on Computer and Communications ICC3*, Chengdu, China, pp. 923-927, 2018.
- [45] Oliveira, B. and Margi, C., "Distributed control plane architecture for software-defined Wireless Sensor Networks" in *IEEE International Symposium on Consumer Electronics ISCE*, Sao Paulo, pp. 85-86, 2016.

- [46] Kobo, H., Abu-Mahfouz, A. and Hancke, G., "A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements" in *IEEE Access*, vol. 5, pp. 1872-1899, 2017.
- [47] Kobo, H., Hancke, G. and Abu-Mahfouz, A., "Towards a distributed control system for software defined Wireless Sensor Networks," in *43rd Annual Conference of the IEEE Industrial Electronics Society IECON 2017*, Beijing, pp. 6125-6130, 2017.
- [48] Kgogo, T., Isong, B. and Abu-Mahfouz, A., "Software defined wireless sensor networks security challenges" in *IEEE AFRICON 2017*, Cape Town, pp. 1508-1513, 2017.
- [49] Jian, D., Chunxiu, X., Muqing, W. and Wenxing, L., "Design and implementation of a novel software-defined wireless sensor network" in *3rd IEEE International Conference on Computer and Communications ICC 2017*, Chengdu, pp. 729-733, 2017.
- [50] Ezdiani, S., Acharyya, I., Sivakumar S. and Al-Anbuky, A., "An IoT Environment for WSN Adaptive QoS," in *IEEE International Conference on Data Science and Data Intensive Systems 2015*, Sydney, NSW, Australia, pp. 586-593, 2015.
- [51] Ezdiani, S., Acharyya, I., Sivakumar S. and Al-Anbuky, A., "An Architectural Concept for Sensor Cloud QoSaaS Testbed," in *Proceedings of the 6th ACM Workshop on Real World Wireless Sensor Networks RealWSN 2015*, pp. 15-18, 2015.
- [52] Modieginyane, K. M., Malekian, R. and Letswamotse, B. B., "Flexible Network Management and Application Service Adaptability in Software Defined Wireless Sensor Networks," in *Journal of Ambient Intelligence and Humanized Computing*, pp. 1621-1630, 2019.
- [53] Ojo, M., Adami, D. and Giordano, S., "A SDN-IoT Architecture with NFV Implementation" in *IEEE Globecom Workshops*, Washington, DC, pp. 1-6, 2016.
- [54] Gupta, G P., "Software-Defined Networking Paradigm in Wireless Sensor Networks," in *Innovations in Software-Defined Networking and Network Functions Virtualization*, IGI Global, pp. 254-267. 2018.
- [55] He, M., Alba, A. M., Basta, A., Blenk, A. and Kellerer, W., "Flexibility in Softwarized Networks: Classifications and Research Challenges" in *IEEE Communications Surveys & Tutorials*, pp. 2600-2636, 2019.
- [56] Fazio, M., Paone, M., Puliafito, A. and Villari, M., "Huge amount of heterogeneous sensed data needs the cloud," in *International Multi-Conference on Systems, Signals & Devices*, Chemnitz, Germany, pp. 1-6, 2012.
- [57] Khan, I., Belqasmi, F., Glitho, R., Crespi, N., Morrow, M. and Polakos, P., "Wireless sensor network virtualization: early architecture and research perspectives," in *IEEE Network*, vol. 29, no. 3, pp. 104-112, 2015.
- [58] Khan, I., Belqasmi, F., Glitho, R., Crespi, N., Morrow, M. and Polakos, P., "Wireless sensor network virtualization: A survey," in *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 553-576, 2016.
- [59] Islam M.M., Hassan M.M., Lee G.W. and Huh E.N., "A survey on virtualization of Wireless Sensor Networks," in *Sensors*, Basel, Switzerland, vol. 12, Issue 2, pp. 2175-2207, 2012.
- [60] Khalid, Z., Fisal, N. and Rozaini, M., "A Survey of Middleware for Sensor and Network Virtualization," in *Sensors*, Vol.14, Issue 12, pp. 24046-24097, 2014.
- [61] Akram, H. and Gokhale, A., "Rethinking the Design of LR-WPAN IoT Systems with Software-Defined Networking," in *2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Washington, DC, USA, pp. 238-243, 2016.
- [62] Van den Abeele, F., Hoebeke, J., Teklemariam, G. K., Moerman, I. and Demeester, P., "Sensor function virtualization to support distributed intelligence in the Internet of things," in *Wireless Personal Communications*, pp. 1415-1436, 2015.

- [63] Li, W., Santos, I., Delicato, F.C., Pires, P.F., Pirmez, L., Wei, W., Song, H., Zomaya, A. and Khan, S., "System modelling and performance evaluation of a three-tier Cloud of Things," in *Future Generation Computer Systems*, 70, pp. 104-125, 2017.
- [64] Yu, Y., Rittle, L. J., Bhandari, V. and LeBrun, J. B., "Supporting concurrent applications in wireless sensor networks" in *Proceedings of the 4th international conference on Embedded networked sensor systems*, pp. 139-152, 2006.
- [65] Khan, I., "Design and analysis of virtualization framework for Wireless Sensor Networks," in *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, Madrid, pp. 1-2, 2013.
- [66] Jayasumana, A. P., Han, Q. and Illangasekare, T. H., "Virtual sensor networks-a resource efficient approach for concurrent applications" in *Fourth International Conference on Information Technology (ITNG'07)*, IEEE, pp. 111-115, 2007.
- [67] Islam, M., Hassan, M. M., Lee, G. W. and Huh, E. N., "A survey on virtualization of wireless sensor networks" in *Sensors*, Vol 12, Issue 2, pp. 2175-2207.
- [68] Garcia A.L. and Tizghadam A., "Application platforms for smart cities," in *2015 IEEE First International Smart Cities Conference (ISC2)*, Guadalajara, pp. 1-6, 2015.
- [69] Wang, X., Wang, J., Zheng, Z., Xu, Y. and Yang, M., "Service Composition in Service-Oriented Wireless Sensor Networks with Persistent Queries," in *6th IEEE Consumer Communications and Networking Conference, CCNC*, pp. 1-5, 2009.
- [70] Baccelli, E., Hahm, O., Günes, M., Wählisch, M. and Schmidt, T., "OS for the IoT - Goals, Challenges, and Solutions," in *Workshop Interdisciplinaire sur la Sécurité Globale (WISG2013)*, Troyes, France, pp. 1-6, 2013.
- [71] Ala-Laurinaho, R., "Sensor data transmission from a physical twin to a digital twin," *Master's Thesis*, 2019.
- [72] Condoluci, M. and Mahmoodi, T., "Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges", in *Computer Networks*, Volume 146, pp. 65-84, 2018.
- [73] Granelli, F., Gebremariam, A.A., Usman, M., Cugini, F., Stamati, V., Alitska, M. and Chatzimisios, "Software defined and virtualized wireless access in future wireless networks: scenarios and standards" in *IEEE Communications Magazine*, Volume 53, Issue 6, pp. 26-34, 2015.
- [74] Stojmenovic, I., "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks", in *2014 Australasian telecommunication networks and applications conference (ATNAC)*, pp. 117-122, 2015.
- [75] Kobo, H. I., Abu-Mahfouz, A. M. and Hancke, G. P., "A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements," in *IEEE Access*, Vol. 5, pp. 182-1899, 2017.
- [76] Ordonez-Lucena, J., Ameigeiras, P., Lopez, D., Ramos-Munoz, J. J., Lorca, J. and Folgueira, J., "Network Slicing for 5G with SDN/NFV: Concepts Architectures and Challenges," in *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80-87, 2017.
- [77] Ding, C. and Shen, L., "Design and Implementation of Programmable Nodes in Software Defined Sensor Networks," in *85th Vehicular Technology Conference (VTC Spring)*, IEEE, pp. 1-5, 2017.
- [78] Fok, C.L., Roman, G.C. and Lu, C., "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pp. 653-662, 2005.
- [79] Egidius, P. M., Abu-Mahfouz, A. M. and Hancke, G. P., "Programmable Node in Software-Defined Wireless Sensor Networks: A Review," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, Washington, DC, USA, pp. 4672-4677, 2018.

- [80] Kipongo, J., Olwal, T. O. and Abu-Mahfouz, A. M., "Topology Discovery Protocol for Software Defined Wireless Sensor Network: Solutions and Open Issues" in *27th International Symposium on Industrial Electronics (ISIE)*, IEEE, pp. 1282–1287, 2018.
- [81] Abdolmaleki, N., Ahmadi, M., Malazi, H. T. and Milardo, S., "Fuzzy topology discovery protocol for SDNbased wireless sensor networks", in *Simulation Modelling Practice and Theory*, Vol. 79, pp. 54–68, 2017.
- [82] Zeng, D., Miyazaki, T., Guo, S., Tsukahara, T., Kitamichi, J. and Hayashi, T., "Evolution of Software-Defined Sensor Networks," in *IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, Dalian, China, pp. 410-413, 2013.
- [83] Miyazaki, T., Yamaguchi, S., Kobayashi, K., Kitamichi, J., Guo, S., Tsukahara, T. and Hayashi, T., "A software defined wireless sensor network," in *2014 International Conference on Computing, Networking and Communications (ICNC)*, Honolulu, HI, USA, pp. 847-852, 2014.
- [84] Nguyen, T. M. C., Hoang, D. B. and Chaczko, Z., "Can SDN Technology Be Transported to Software- Defined WSN/IoT?" in *2016 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Chengdu, pp. 234-239, 2016.
- [85] Kobo, H. I., Abu-Mahfouz, A. M. and Hancke, G. P., "A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements" in *IEEE Access*, Vol. 5, pp. 1872–1899, 2017.
- [86] Mostafaei, H. and Menth, M., "Software-defined wireless sensor networks: A survey" in *Journal of Network and Computer Applications*, Vol. 119, pp. 42–56, 2018.
- [87] Bera, S., Misra, S. and Vasilakos, A. V., "Software- Defined Networking for Internet of Things: A Survey", in *IEEE Internet of Things Journal*, Vol. 4, Issue 6, pp. 1994-2008, 2017.
- [88] Haque, I., Nurujjaman, M., Harms, J. and Abu-Ghazaleh, N., "SDSense: An Agile and Flexible SDN-Based Framework for Wireless Sensor Networks" in *IEEE Transactions on Vehicular Technology*, vol. 68, Issue 2, pp. 1866– 1876, 2019.
- [89] Jemal, A. and Halima, R. B., "A QoS-driven Self-Adaptive Architecture for Wireless Sensor Networks" in *Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Hammamet, Tunisia, pp. 125–130, 2013.
- [90] Galluccio, L., Milardo, S., Morabito, G. and Palazzo, S., "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for Wireless Sensor networks" in *IEEE Conference on Computer Communications (INFOCOM)*, pp. 513–521, 2015.
- [91] Bera, S., Misra, S., Roy, S. K. and Obaidat, M. S., "Soft-WSN: Software-Defined WSN Management System for IoT Applications" in *IEEE Systems Journal*, vol. 12, Issue 3, 2074–2081, 2018.
- [92] Theodorou, T. and Mamatas, L., "Software defined topology control strategies for the Internet of Things" in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFVSDN)*, pp. 236–241, 2017.
- [93] Luo, T., Tan, H.P. and Quek, T.Q., "Sensor OpenFlow: Enabling software-defined wireless sensor networks," in *IEEE Communications letters*, vol. 16, Issue 11, pp. 1896-1899, 2012.
- [94] Mahmud, A., Rahmani, R. and Kanter, T., "Deployment of flow-sensors in Internet of things' virtualization via openflow," in *Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing (MUSIC)*, pp. 195-200, 2012.
- [95] Mouradian, C., Saha, T., Sahoo, J., Glitho, R., Morrow, M. and Polakos, P., "NFV based gateways for virtualized wireless sensor networks: A case study" in *IEEE*

- International Conference on Communication Workshop (ICCW)*, London, pp. 1883-1888, 2015.
- [96] Mouradian, C., Saha, T., Sahoo, J., Abu-Lebdeh, M., Glitho, R., Morrow, M. and Polakos, P, "Network functions virtualization architecture for gateways for virtualized wireless sensor and actuator networks" in *IEEE Network*, vol. 30, no. 3, pp. 72-80, 2016.
- [97] Luo, S., Wang, H., Wu, J., Li, J., Guo, L. and Pei, B., "Improving Energy Efficiency in Industrial Wireless Sensor Networks Using SDN and NFV" in *IEEE 83rd Vehicular Technology Conference (VTC Spring)*, Nanjing, 2016.
- [98] Jacobsson, M. and Orfanidis, C., "Using software-defined networking principles for wireless sensor networks," in *11th Swedish National Computer Networking Workshop (SNCNW)*, Karlstad, pp. 2015
- [99] Szilvási, S., Babják, B., Völgyesi, P. and Lédeczi, A., "Marmote SDR: Experimental platform for low-power wireless protocol stack research," in *Journal of Sensor and Actuator Networks*, vol. 2, pp. 631-652, 2013.
- [100] De Oliveira, B.T., Gabriel, L.B. and Margi, C.B., "TinySDN: Enabling multiple controllers for software-defined wireless sensor networks," in *IEEE Latin America Transactions*, vol. 13, Issue 11, pp. 3690-3696, 2015.
- [101] Österlind, F., "A sensor network simulator for the Contiki OS" in *SICS Research Report*, 2006.
- [102] Dunkels, A., Gronvall, B. and Voigt, T., "Contiki-a lightweight and flexible operating system for tiny networked sensors" in *29th annual IEEE international conference on local computer networks*, IEEE, pp. 455-462, 2004.
- [103] Belonovsky, A.V., Makukha, V.K., Markov, A.V. and Shapovalov, S., "Development of low-power device for wireless data transmission under the 6LoWPAN standard," in *14th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices*, Novosibirsk, Russia, pp. 76-78, 2013.
- [104] Dutta, P. and Dunkels, A., "Operating systems and network protocols for wireless sensor networks," in *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, pp. 68-84, 2012.
- [105] Dunkels, A., Schmidt, O., Voigt, T. and Ali, M., "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, pp. 29-42, 2006.
- [106] Dunkels, A., "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, Vol. 44, 2007.
- [107] Mitton, N., Papavassiliou, S., Puliafito, A. and Trivedi, K. S., "Combining Cloud and sensors in a smart city environment," in *EURASIP Journal on Wireless Communications and Networking*, 2012.
- [108] Reusing, T., "Comparison of operating systems tinyos and contiki," in *Sens. Nodes-Operation, Netw. Appli.(SN)*, vol. 7., 2012.
- [109] Ahlsén, M., "Enabling the Business-Based Internet of Things and Services," in *D2.2.1 Technology Watch Report.*, 2010.
- [110] Granlund, M. and Hoppe, C., "Evaluating the functionality of an Industrial Internet of Things system in the Fog," 2018.
- [111] Ellmer, C., "Openthread vs. contiki ipv6: An experimental evaluation," 2017.
- [112] Roussel, K. and Song, Y. Q., "A critical analysis of Contiki's network stack for integrating new MAC protocols," (*Doctoral dissertation*), INRIA Nancy, 2013.
- [113] [online] Available: <https://www.ti.com/lit/ds/symlink/cc2538.pdf>.
- [114] Sadasivan, S., "An introduction to the arm cortex-m3 processor," in *ARM*, 2006.

- [115] Venkatesh, G, "Semiconductor solutions for the Internet of things: The role of event detection, asynchronous design, energy harvesting and flexible electronics," in *Journal of the Indian Institute of Science*, pp. 441-462, 2013.
- [116] Maksimović, M., Vujović, V., Davidović, N., Milošević, V. and Perišić, B., "Raspberry Pi as Internet of things hardware: performances and constraints." in *design issues*, 2014.
- [117] Barbato, A., M. Barrano, A. Capone and N. Figiani, "Resource Oriented and Energy Efficient Routing Protocol for IPv6 Wireless Sensor Networks", in *2013 IEEE Online Conference on Green Communications (OnlineGreenComm)*, pp. 163-168, 2013.
- [118] Dunkels, A., Gronvall, B. and Voigt, T., "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *29th annual IEEE international conference on local computer networks*, pp. 455-462, 2004.
- [119] Sundani, H., Li, H., Devabhaktuni, V., Alam, M. and Bhattacharya, P., "Wireless sensor network simulators a survey and comparisons." in *International Journal of Computer Networks (IJCN)*, vol. 2, Issue 5, pp. 249-265, 2011.
- [120] Li, Y. and Chen, M., "Software-defined network function virtualization: A survey," in *IEEE Access*, vol. 3, pp. 2542-2553, 2015.
- [121] Condoluci, M. and Mahmoodi, T., "Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges" in *Computer Networks*, vol. 146, pp. 65-84.
- [122] Salahuddin, M. A., Al-Fuqaha, A., Guizani, M., Shuaib, K. and Sallabi, F., "Softwarization of Internet of things infrastructure for secure and smart healthcare," in *arXiv preprint arXiv:1805.11011*, 2018.
- [123] He, M., Alba, A. M., Basta, A., Blenk, A. and Kellerer, W., "Flexibility in softwarized networks: Classifications and research challenges," in *IEEE Communications Surveys & Tutorials*, vol. 21, Issue 3, pp. 2600-2636, 2019.
- [124] [online] Available: https://en.wikipedia.org/wiki/Transmitter_power_output.
- [125] Lavratti, F., Ceratti, A., Prestes, D., Pinto, A.R., Bolzani, L., Vargas, F., Montez, C., Hernandez, F., Gatti, E. and Silva, C. "A Transmission Power Self-Optimization Technique for Wireless Sensor Networks," in *International Scholarly Research Notices*, 2012.
- [126] [online] Available: <https://www.betasolutions.co.nz/blog/low-power-firmware-design-for-wireless-sensor-networks>
- [127] Dias, G. M., Nurchis, M. and Bellalta, B., "Adapting sampling interval of sensor networks using on-line reinforcement learning," in *IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 460-465, 2016.
- [128] Senouci, M. R., and Mellouk, A., "Deploying wireless sensor networks: theory and practice," in *Elsevier*, 2016.
- [129] Jayaram, K., Janani, K., Jeyaguru, R., Kumaresh, R. and Muralidharan, N., "Forest Fire Alerting System With GPS Co-ordinates Using IoT," in *5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, IEEE, pp. 488-491, 2019.
- [130] Jaiswal, R. K., Mukherjee, S., Raju, K. D. and Saxena, R., "Forest fire risk zone mapping from satellite imagery and GIS," in *International Journal of Applied Earth Observation and Geoinformation*, vol. 4, Issue 1, pp. 1-10, 2002.
- [131] Pradhan, B., Suliman, M. D. H. B. and Awang, M. A. B., "Forest fire susceptibility and risk mapping using remote sensing and geographical information systems (GIS)," in *Disaster Prevention and Management: An International Journal*, 2007.
- [132] Kadir, E. A., Rosa, S. L. and Yulianti, A., "Application of WSNs for detection land and forest fire in Riau Province Indonesia," in *2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*, IEEE, pp. 25-28, 2018.

- [133] Grover, K., Kahali, D., Verma, S. and Subramanian, B., "WSN-based system for forest fire detection and mitigation," in *Emerging Technologies for Agriculture and Environment*, Springer, Singapore, pp. 249-260, 2020.
- [134] Sinha, D., Kumari, R. and Tripathi, S., "Semisupervised classification based clustering approach in WSN for forest fire detection" in *Wireless Personal Communications*, pp. 2561-2605, 2019.
- [135] Lei, Z., and Lu, J., "Distributed coverage of forest fire border based on WSN" in *2010 2nd International Conference on Industrial and Information Systems*, vol. 1, IEEE, pp. 341-344, 2010.
- [136] Xu, Y. H., Sun, Q. Y. and Xiao, Y. T., "An environmentally aware scheme of wireless sensor networks for forest fire monitoring and detection" in *Future Internet*, p. 102, 2018.
- [137] Zhang, J., Li, W., Han, N., and Kan, J., "Forest fire detection system based on a ZigBee wireless sensor network" in *Frontiers of Forestry in China*, pp. 369-374, 2008.
- [138] Díaz-Ramírez, A., Tafoya, L. A., Atempa, J. A. and Mejía-Alvarez, P., "Wireless sensor networks and fusion information methods for forest fire detection," in *Procedia Technology*, 3, pp. 69-79. 2012.
- [139] Aslan, Y. E., Korpeoglu, I. and Ulusoy, Ö., "A framework for use of wireless sensor networks in forest fire detection and monitoring," in *Computers, Environment and Urban Systems*, vol. 36, Issue 6, pp.614-625, 2012.
- [140] Bayo, A., Antolín, D., Medrano, N., Calvo, B., and Celma, S., "Early detection and monitoring of forest fire with a wireless sensor network system," in *Procedia Engineering*, 5, pp. 248-251, 2010.
- [141] Trivedi, K. and Srivastava, A. K., "An energy efficient framework for detection and monitoring of forest fire using mobile agent in wireless sensor networks," in *2014 IEEE International Conference on Computational Intelligence and Computing Research*, pp. 1-4, IEEE, 2014.
- [142] Yu, L., Wang, N., and Meng, X., "Real-time forest fire detection with wireless sensor networks," in *Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing*, vol. 2, pp. 1214-1217. IEEE, 2005.
- [143] Lloret, J., Garcia, M., Bri, D. and Sendra, S., "A wireless sensor network deployment for rural and forest fire detection and verification" in *sensors*, 9(11), pp. 8722-8747, 2009.
- [144] Anand, S., & RK, K. M., "FPGA implementation of artificial neural network for forest fire detection in wireless sensor network," in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, pp. 265-270, IEEE, 2017.
- [145] Gilani, M. H. S., Sarrafi, I. and Abbaspour, M., "An adaptive CSMA/TDMA hybrid MAC for energy and throughput improvement of wireless sensor networks," in *Ad Hoc Networks*, vol. 11, Issue 4, pp. 1297-1304, 2013.
- [146] <https://hewlettpackard.github.io/wireless-tools/Linux.Wireless.mac.html>.
- [147] Rubio, B., Diaz, M. and Troya, J. M., "Programming approaches and challenges for wireless sensor networks" in *2007 Second International Conference on Systems and Networks Communications (ICSNC 2007)*, pp. 36-36, IEEE, 2007.
- [148] Mohapatra, S. and Khilar, P. M., "Forest fire monitoring and detection of faulty nodes using wireless sensor network," in *IEEE Region 10 Conference (TENCON)*, Singapore, pp. 3232-3236, 2016.
- [149] Sonule, M. G. and Nikam, S., "Role of sensor virtualization in wireless sensor networks," in *International Journal of Computer Science and Information Technologies*, vol. 5 (1), pp. 699-703, 2014.

- [150] Kaur, H., Sood, S.K. and Bhatia, M., "Cloud-assisted green IoT-enabled comprehensive framework for wildfire monitoring," *Cluster Computing, Springer*, pp.1-14, 2019.
- [151] Al-Hamid, D. Z. and Al-Anbuky, A., "Vehicular grouping and network formation: virtualization of network self-healing," in *International Conference on Internet of Vehicles, Springer, Cham*, pp. 106-121, 2018.
- [152] Jacobsson, M. and Orfanidis, C., "Using software-defined networking principles for wireless sensor networks," in *SNCNW 2015, Karlstad, Sweden, 2015*.
- [153] Nevala, C., "Mobility management for software defined wireless sensor networks," 2016

Appendix

A.1 Example Integrated Firmware (Integrated Code Consisting of Leaf and Router Modules)

```

#include "contiki.h"
#include "sys/etimer.h"
#include "sys/rtimer.h"
#include "dev/leds.h"
#include "dev/button-sensor.h"
#include "dev/watchdog.h"
#include "dev/serial-line.h"
#include "net/rime/broadcast.h"
#include <stdio.h>
#include <stdint.h>
#include <math.h>

/*-----*/
#define LOOP_INTERVAL    CLOCK_SECOND
#define LEDS_OFF_HYSTERISIS (RTIMER_SECOND >> 1)
#define LEDS_PERIODIC    LEDS_YELLOW
#define LEDS_BUTTON      LEDS_RED
#define LEDS_SERIAL_IN   LEDS_ORANGE
#define LEDS_REBOOT      LEDS_ALL
#define LEDS_RF_RX       (LEDS_YELLOW | LEDS_ORANGE)
#define BROADCAST_CHANNEL 129
#define N_DECIMAL_POINTS_PRECISION (100)

#define MAX_NODES        3
#define TIME_IN_SEC      47
/*-----*/

```

```

static struct etimer et;
static struct rtimer rt;
static uint16_t counter;
static uint16_t node_ID = 1; //Address 0xCD, 0xCD

int Transmit_Flag = 0;
int i = 0;
static uint16_t a[3];
float light_dbl;
float temp_dbl;
float rssi_dbl;
int lightintpart;
int lightdecpart;
int tempintpart;
int tempdecpart;
int rssiintpart;
int rssiidecpart;
static uint16_t c[7];
int existing_function_flag=0;
static uint16_t function_change_counter=0;

/*****DECLARATION OF ROUTER VARIABLES*****/
static uint16_t counter;
static uint16_t count_flag;
short signed light[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed temperature[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed rssi[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed lightdec[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed temperaturedec[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed rssiidec[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

```

```

short signed values_to_be_transmitted[7];

int x;

int y;

/*****DECLARATION OF ROUTER VARIABLES*****/

/*-----*/

PROCESS(cc2538_demo_process, "cc2538 demo process");
AUTOSTART_PROCESSES(&cc2538_demo_process);

/*-----*/

static void
broadcast_rcv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    if(existing_function_flag==0){
        uint16_t *dataptr_temp;

        dataptr_temp= (uint16_t *)packetbuf_dataptr();
        if(dataptr_temp[0] == node_ID)
        {
            Transmit_Flag=1;
            a[0]=dataptr_temp[0];
            a[1]=dataptr_temp[1];
            a[2]=dataptr_temp[2];
        }
        else
        {
            Transmit_Flag=0;
        }
    }

    if(existing_function_flag==1){
        int16_t *dataptr_temp1;

```



```

    dataptr_temp1= (int16_t *)packetbuf_dataptr();
//if (dataptr_temp1[0]==a[0]){
    //for (x=1;x<=15;x++) {
        //if (a[0]==x) {
//for (dataptr_temp1[0]=x;dataptr_temp1[0]<=x;dataptr_temp1[0]++){
    x =dataptr_temp1[0];
    light[x] =dataptr_temp1[1];
    temperature[x] =dataptr_temp1[2];
    rssi[x] =dataptr_temp1[3];
    lightdec[x] =dataptr_temp1[4];
    temperaturedec[x] =dataptr_temp1[5];
    rssi_dec[x] =dataptr_temp1[6];
}
}
/*-----*/
static const struct broadcast_callbacks bc_rx = { broadcast_recv };
static struct broadcast_conn bc;
/*-----*/
void
rt_callback(struct rtimer *t, void *ptr)
{
    leds_off(LED_PERIODIC);
}
/*-----*/
PROCESS_THREAD(cc2538_demo_process, ev, data)
{

    PROCESS_EXITHANDLER(broadcast_close(&bc))

    PROCESS_BEGIN();

```

```

if(existing_function_flag==0){
    counter = 0;
    broadcast_open(&bc, BROADCAST_CHANNEL, &bc_rx);
}

if(existing_function_flag==1){
    counter = 0;
    count_flag=0;
    broadcast_open(&bc, BROADCAST_CHANNEL, &bc_rx);
}

while(1) {
if(existing_function_flag==1){
    if(count_flag==0)
    {
    etimer_set(&et, CLOCK_SECOND);
    count_flag=1;
    }
}

if(existing_function_flag==0){
    etimer_set(&et, CLOCK_SECOND*1);
}

    PROCESS_YIELD();
//if(Transmit_Flag==1)
//{          //leds_toggle(LED_RF_RX);
if(existing_function_flag==0){
    /***** FOR
LIGHT*****/

    light_dbl=rand();
    lightintpart = (int)light_dbl;
    lightintpart = abs(lightintpart);

```

```

lightintpart= (lightintpart % 27995);
lightintpart= (lightintpart+5);
lightdecpart
((int)(light_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISI
ON);
lightdecpart = abs(lightdecpart);

/*****
*****/

/*****FORTEMPERATURE*****/

temp_dbl = rand();
tempintpart = (int)temp_dbl;
tempintpart=abs(tempintpart);
tempintpart = (tempintpart % 35);
tempintpart = (tempintpart+5);
tempdecpart
((int)(temp_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISI
ON);
tempdecpart=abs(tempdecpart);

/*****
*****/

/*****FORRSSI*****/

rssi_dbl=rand();
rssiintpart = (int)rssi_dbl;
rssiintpart=abs(rssiintpart);
rssiintpart= (rssiintpart % 88);
rssiintpart= (rssiintpart + 10);
rssiintpart=rssiintpart*(-1);

```

```

        rssidecpart
((int)(rssi_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISIO
N);

        rssidecpart=abs(rssidecpart);

        rssidecpart=0;

/*****
*****/

        c[0]=node_ID;
        c[1]=lightintpart;
        c[2]=tempintpart;
        c[3]=rssiintpart;
        c[4]=lightdecpart;
        c[5]=tempdecpart;
        c[6]=rssidecpart;

        if(Transmit_Flag==1)
{
        printf("This is end device with node_ID=%d transmitting the following values to
the Coordinator.\n", node_ID);

        printf("1. Ambient raw light = '%d.%d' lux.\n", lightintpart, lightdecpart);

        printf("2. Temperature value = '%d.%d' degree celsius.\n",tempintpart
,tempdecpart);

        printf("3. rssi      = '%d.%d' dBm.\n",rssiintpart, rssidecpart);

        printf("Func_ch_count      = '%d.\n",function_change_counter);

        packetbuf_copyfrom(&c, sizeof(c));

        broadcast_send(&bc);

        if(function_change_counter>=50)
        {existing_function_flag=1;
        }

        function_change_counter++;

        Transmit_Flag=0;

```

```

        if(a[2]==1)
        {
etimer_set(&et, CLOCK_SECOND*a[1]);
        a[2]=0;
        }
        if(ev == PROCESS_EVENT_TIMER && a[0]!=node_ID)
        {
etimer_set(&et, CLOCK_SECOND*1);
        }
}
}

```

```

if(existing_function_flag==1){
if(ev == PROCESS_EVENT_TIMER) {
    leds_on(LED_PERIODIC);

    counter++;
printf("\nCounter value = %d counts.", counter);
    a[0]=counter;
        a[1]=TIME_IN_SEC - 45;
    a[2]=1;
etimer_set(&et, CLOCK_SECOND*3);
packetbuf_copyfrom(&a, sizeof(a));
broadcast_send(&bc);
//printf("node_ID=%d\n", a[0]);
//if (a[0]== 15) {
//printf("\n\n");

//}
if(counter==51)

```

```

        {
            counter=(counter % 51);
            printf("\nValues reported by the end devices (to the router) are as follows:");
for (x=1;x<=4;x++)
    {
//print node lines

printf("\n|Node %d: %d,%d.%d,%d|", x, light[x], temperature[x], temperaturedec[x], rssi[x]);
values_to_be_transmitted[0]=x;
values_to_be_transmitted[1]=light[x];
values_to_be_transmitted[2]=temperature[x];
values_to_be_transmitted[3]=rssi[x];
values_to_be_transmitted[4]=lightdec[x];
values_to_be_transmitted[5]=temperaturedec[x];
values_to_be_transmitted[6]=rssi_dec[x];

packetbuf_copyfrom(&values_to_be_transmitted, sizeof(values_to_be_transmitted));
broadcast_send(&bc);
    }
printf("\n");
        etimer_set(&et, CLOCK_SECOND*TIME_IN_SEC);
    }

}

}

}
PROCESS_END();
}

```

A.2 Codes for Transmitter and Receiver Nodes (Reg. ‘Arrival’ and ‘Service’ rates of End-device and Coordinator nodes respectively)

The code for the ‘end device’ or ‘transmitter’ node is provided below.

```

#include "contiki.h"
#include "cpu.h"
#include "sys/etimer.h"
#include "sys/rtimer.h"
#include "dev/leds.h"
#include "dev/uart.h"
#include "dev/button-sensor.h"
#include "dev/adc-sensor.h"
#include "dev/watchdog.h"
#include "dev/serial-line.h"
#include "dev/sys-ctrl.h"
#include "net/rime/broadcast.h"
#include <stdio.h>
#include <stdint.h>
#include <math.h>
/*-----*/
#define LOOP_INTERVAL    CLOCK_SECOND
#define LEDS_OFF_HYSTERISIS (RTIMER_SECOND >> 1)
#define LEDS_PERIODIC    LEDS_YELLOW
#define LEDS_BUTTON      LEDS_RED
#define LEDS_SERIAL_IN   LEDS_ORANGE
#define LEDS_REBOOT      LEDS_ALL
#define LEDS_RF_RX       (LEDS_YELLOW | LEDS_ORANGE)
#define BROADCAST_CHANNEL 129
#define N_DECIMAL_POINTS_PRECISION (100)
/*-----*/

```

```
static struct etimer et;
//static struct etimer et;
//static struct etimer ed;
static struct rtimer rt;
static uint16_t counter;
static uint16_t node_ID = 1; //Address 0xCD, 0xCD
//static uint16_t node_ID = 2; //Address 0xCD, 0xCE
//static uint16_t node_ID = 3; //Address 0xCD, 0xCF
//static uint16_t node_ID = 4; //Address 0xCD, 0xDA
//static uint16_t node_ID = 5; //Address 0xCD, 0xDB
//static uint16_t node_ID = 6; //Address 0xCD, 0xDC
//static uint16_t node_ID = 7; //Address 0xCD, 0xDD
//static uint16_t node_ID = 8; //Address 0xCD, 0xDE
//static uint16_t node_ID = 9; //Address 0xCD, 0xDF
//static uint16_t node_ID = 10; //Address 0xCD, 0xEA
//static uint16_t node_ID = 11; //Address 0xCD, 0xEB
//static uint16_t node_ID = 12; //Address 0xCD, 0xEC
//static uint16_t node_ID = 13; //Address 0xCD, 0xED
//static uint16_t node_ID = 14; //Address 0xCD, 0xEE
//static uint16_t node_ID = 15; //Address 0xCD, 0xEF

int Transmit_Flag = 0;
int i = 0;
int j;
static uint16_t a[3];
float light_dbl;
int16_t temp;
float temp_dbl;
signed short rssi;
float rssi_dbl;
```



```

int lightintpart;
int lightdecpart;
//int lightdecpart_int;
int tempintpart;
int tempdecpart;
//int tempdecpart_int;
int rssiintpart;
int rssiidecpart;
//int rssiidecpart_int;
//static uint16_t c[2];
//static uint16_t d[2];
static uint16_t c1;
static uint16_t d1;
int Recv_Flag=0;
uint16_t counterlighttr=0;
uint16_t countertemptr=0;

//int tx;
//long int countincr=0;
/*-----*/
PROCESS(cc2538_demo_process, "cc2538 demo process");
AUTOSTART_PROCESSES(&cc2538_demo_process);
/*-----*/
static void
broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    uint16_t *dataptr_temp='\0';

    dataptr_temp= (uint16_t *)packetbuf_dataptr();
    if(dataptr_temp[0] == node_ID)

```

```
{
    Transmit_Flag=1;
    a[0]=dataptr_temp[0];
    a[1]=dataptr_temp[1];
    a[2]=dataptr_temp[2];
}
else
{
    Transmit_Flag=0;
}
}

//if(*dataptr_temp!= '\0'){
//Recv_Flag=1;
//}
//else
//{
//Recv_Flag=0;
//}
//{
//Transmit_Flag=1;
//a[0]=dataptr_temp[0];
//a[1]=dataptr_temp[1];
//a[2]=dataptr_temp[2];
//}
//else
//{
//Transmit_Flag=0;
//}
//}
```

```

/*-----*/
static const struct broadcast_callbacks bc_rx = { broadcast_recv };
static struct broadcast_conn bc;
/*-----*/

void
rt_callback(struct rtimer *t, void *ptr)
{
    leds_off(LED_PERIODIC);
}
/*-----*/

PROCESS_THREAD(cc2538_demo_process, ev, data)
{

    PROCESS_EXITHANDLER(broadcast_close(&bc))

    PROCESS_BEGIN();

    counter = 0;

    //count=0;

    broadcast_open(&bc, BROADCAST_CHANNEL, &bc_rx);

    while(1)
    {
        etimer_set(&et, CLOCK_SECOND*(0.025)); //Case 1
        //etimer_set(&et_2, CLOCK_SECOND*(0.2));
        //etimer_set(&et, CLOCK_SECOND*1); //Case 2
        //etimer_set(&et, CLOCK_SECOND*10); //Case 3
    }
}

```

```
//etimer_set(&et, CLOCK_SECOND*1);  
PROCESS_YIELD();  
counter++;  
  
//if(Transmit_Flag==1)  
//{          //leds_toggle(LED_S_RF_RX);  
            light_dbl=adc_sensor.value(ADC_SENSOR_ALS);  
            if (light_dbl<=4500){  
light_dbl= light_dbl*0;  
            }  
            if ((light_dbl>=4501) && (light_dbl<=9516)){  
light_dbl= ((light_dbl - 4500)/(627/5));  
            }  
            if ((light_dbl>=9517) && (light_dbl<=11572)){  
light_dbl= ((light_dbl - 5404)/102.8);  
            }  
            if ((light_dbl>=11573) && (light_dbl<=15744)){  
light_dbl= ((light_dbl - 11475)/ (1043/58));  
            }  
            if ((light_dbl>=15745) && (light_dbl<=16780)){  
light_dbl= ((light_dbl - 14462)/(259/59));  
            }  
            if ((light_dbl>=16781) && (light_dbl<=20000)){  
light_dbl= ((light_dbl - 16196)/(805/728));  
            }  
            if ((light_dbl>=20001) && (light_dbl<=21772)){  
light_dbl= ((light_dbl - 19616)*9.28);  
            }  
            if ((light_dbl>21773) && (light_dbl<=25116)){  
light_dbl= ((light_dbl - 12483)*2.153);
```

```

}
if ((light_dbl>25117) && (light_dbl<=25184)){
light_dbl= ((light_dbl - 24730)*70.59);
}

    lightintpart = (int)light_dbl;
    lightdecpart
((int)(light_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISI =
ON);

temp = adc_sensor.value(ADC_SENSOR_TEMP);
//temp_dbl = (((25 + ((temp >> 4) - 1422) * 10 / 42) - 3)/2);
//temp_dbl=(((temp >> 4) - 1422)*0.2381)+22);

//*****/
/

    //For Node_IDs = 2,5,6 and 10
//temp_dbl=((((temp >> 4) - 1422)*0.2381)+22)/2);
//temp_dbl=temp_dbl + 2.56;

//*****/
/

//*****/
/

    //For Node_IDs = 8 and 9
//temp_dbl=(((temp >> 4) - 1422)*0.2381)+22);
//temp_dbl=temp_dbl-4;

```

```
/**  
/  

```

```
/**  
/  

```

```
    //For Node_IDs = 14 and 11  
    //temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
```

```
/**  
/  

```

```
/**  
/  

```

```
    //For Node_IDs = 12, 13 and 3  
    //temp_dbl((((temp >> 4) - 1422)*0.2381)+22);  
    //temp_dbl=temp_dbl-1.61;
```

```
/**  
/  

```

```
/**  
/  

```

```
    //For Node_IDs = 1 and 15  
    //temp_dbl((((temp >> 4) - 1422)*0.2381)+22);  
    //temp_dbl=temp_dbl + 1.43;
```

```
/**  
/  

```

```
/**  
/  

```

```

//For Node_IDs = 7
//temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
//temp_dbl=temp_dbl - 6.9;

//*****/
/

//*****/
/

//For Node_IDs = 4
temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
temp_dbl=temp_dbl - 5.19;

//*****/
/

tempintpart = (int)temp_dbl;
tempdecpart =
((int)(temp_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISI
ON);

if (tempdecpart<0) {
tempdecpart = tempdecpart * (-1);
}

rssi=packetbuf_attr(PACKETBUF_ATTR_RSSI);
rssi_dbl=rssi;
rssiintpart = (int)rssi_dbl;
rssiidecpart =
((int)(rssi_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISIO
N);

/*
if (counter==80){
counter=counter%80;

```

```
tx = 4321;
}
else {
tx = 0;
}
*/

c1=lightintpart;
d1=tempintpart;
struct transmit_data {
char type;
uint16_t data1;
uint16_t data2;
uint16_t data3;
};
// struct transmit_data data_val;
struct transmit_data data_val_light;
data_val_light.type='L';
data_val_light.data1=c1;
data_val_light.data2=node_ID;
data_val_light.data3=counterlightr;

struct transmit_data data_val_temp;
data_val_temp.type='T';
data_val_temp.data1=d1;
data_val_temp.data2=node_ID;
data_val_temp.data3=countertemptr;

//if(Recv_Flag==1){
//etimer_set(&ed, CLOCK_SECOND*(1/40));
```



```

        //Recv_Flag=0;
    //}

    // struct transmit_data data_val;

    //if(counter%1==0){
    //if(counter%40==0){
    if (counter%5==0){
        counterlightr++;

        printf("This is end device with node_ID=%d transmitting the following values to
the Coordinator.\n", node_ID);

        printf("1. Ambient raw light = '%d' lux.\n", lightintpart);
        printf(" Counterlightr = '%d'\n", counterlightr);

        packetbuf_copyfrom(& data_val_light, sizeof(data_val_light));
        broadcast_send(&bc);
        //Recv_Flag=0;
    }

    if(counter%8==0){
        //if(etimer_expired(&et_2)){
        countertemptr++;

        printf("This is end device with node_ID=%d transmitting the following values to
the Coordinator.\n", node_ID);

        printf("2. Temperature value = '%d' degree celsius.\n", tempintpart);
        printf(" Countertemptr = '%d'\n", countertemptr);

        //count++;

        //d[0]=count;

```

```
packetbuf_copyfrom(& data_val_temp, sizeof(data_val_temp));
broadcast_send(&bc);
    //Recv_Flag=0;
}

//if(counter%13==0)
    //{
    //printf("\n\nSet of 13 values\n\n");
    //}

Transmit_Flag=0;
    /*if(a[2]==1)
    {
etimer_set(&et, CLOCK_SECOND*a[1]);
a[2]=0;
    }
    if(ev == PROCESS_EVENT_TIMER && a[0]!=node_ID)
    {
etimer_set(&et, CLOCK_SECOND*1);
    }
    */
//}

}
PROCESS_END();
}
```

The code for the ‘coordinator’ or ‘receiver’ node is provided below.

```

#include "contiki.h"
#include "cpu.h"
#include "sys/etimer.h"
#include "sys/rtimer.h"
#include "dev/leds.h"
#include "dev/uart.h"
#include "dev/button-sensor.h"
#include "dev/adc-sensor.h"
#include "dev/watchdog.h"
#include "dev/serial-line.h"
#include "dev/sys-ctrl.h"
#include "net/rime/broadcast.h"
#include <stdio.h>
#include <stdint.h>
#include <math.h>

/*-----*/
#define LOOP_INTERVAL          CLOCK_SECOND
#define LEDS_OFF_HYSTERISIS    (RTIMER_SECOND >> 1)
#define LEDS_PERIODIC         LEDS_YELLOW
#define LEDS_BUTTON           LEDS_RED
#define LEDS_SERIAL_IN        LEDS_ORANGE
#define LEDS_REBOOT           LEDS_ALL
#define LEDS_RF_RX            (LEDS_YELLOW | LEDS_ORANGE)
#define BROADCAST_CHANNEL     129
#define ARRAY_LENGTH_RT_LIGHT  20// Original value was 27.
#define ARRAY_LENGTH_DT_TEMP   20// Original value was 69.
#define N_DECIMAL_POINTS_PRECISION (100)

```

```
struct transmit_data {
    char type;
    uint16_t data1;
    uint16_t data2;
    uint16_t data3;
};

struct transmit_data *data_val_rcv;

/*-----*/
/*Coordinator address 0xCD, 0xCC*/
/*-----*/

static struct etimer et_1;
static struct etimer et_2;

int etimercounter;

int i;
int x;
int y;
int z;

int ser_countl=0;
int lost_countl=0;
int ser_countt=0;
int lost_countt=0;

int total_countl;
int total_countt;

signed int lightcounttotal=0;
signed int tempcounttotal=0;
signed int lightcounttotal_exit=0;
signed int tempcounttotal_exit=0;

int lightcounttotal_unabletoexit_thatis_lost;
int lightcountwrwb=0;
```

```

int tcount=0;
int totcount;
float lightcounttotal_temp = 0.0;
float lightcounttotal_temp_exit=0.0;

float buffer_usage_real_time_light;
float buffer_usage_temp_real_time_light;
float buffer_usage_dec_real_time_light;
int buffer_usage_decpart_real_time_light;
int buffer_usage_intpart_real_time_light;

float buffer_usage_real_time_temp;
float buffer_usage_temp_real_time_temp;
float buffer_usage_dec_real_time_temp;
int buffer_usage_decpart_real_time_temp;
int buffer_usage_intpart_real_time_temp;

int bufsizeinstantlintpart;
int bufsizeinstantldecpart;
int bufsizeinstantt;
int lightlosscount;
int templosscount;
long int indexyl=ARRAY_LENGTH_RT_LIGHT;
long int indexyt=ARRAY_LENGTH_DT_TEMP;
uint16_t arrlight[ARRAY_LENGTH_RT_LIGHT];
uint16_t arrtemp[ARRAY_LENGTH_DT_TEMP];
uint16_t *Light;
uint16_t *Temp;
uint16_t Node_ID;
uint16_t counterlight;

```

```

uint16_t counterlight_temp;
uint16_t countertemp;
uint16_t countertemp_temp;
uint16_t countertemp;
signed int tpcinbuf;
signed int tpcinbuf_temp;
signed int packetlosstemp;
signed int lpcinbuf;
signed int lpcinbuf_temp;
signed int packetlosslight;
long int serviceratebufferlight[ARRAY_LENGTH_RT_LIGHT];
long int serviceratebuffertemp[ARRAY_LENGTH_DT_TEMP];
long int lostpacketslight[ARRAY_LENGTH_RT_LIGHT];
long int lostpacketstemp[ARRAY_LENGTH_DT_TEMP];
int lassign=0;
int tassign=0;
int buffill;
/*-----*/
PROCESS(cc2538_demo_process, "cc2538 demo process");
AUTOSTART_PROCESSES(&cc2538_demo_process);
/*-----*/
static void
broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)
{
data_val_recv= (struct transmit_data*)packetbuf_dataptr();

if(data_val_recv -> type=='L')
{
//lcount++;
lightcounttotal++;

```

```

//bufsizeinstantl=((ARRAY_LENGTH_RT_LIGHT)-lcount);
Light = &data_val_recv->data11;
Node_ID=data_val_recv->data2;
counterlight=data_val_recv->data3;
counterlight_temp = counterlight;
//printf("\nReceived light packet size:      %d      Index:      %li",packetbuf_datalen(),
indexyl);
printf("\nReceived light packets count:      %d",  lightcounttotal);
printf("\nNode ID of end device =      %d",  Node_ID);
//printf("\nCounterlight received =      %d",  counterlight);
printf("\n-----");
}
else if(data_val_recv -> type=="T"){
tempcounttotal++;
Temp = &data_val_recv->data11;
Node_ID=data_val_recv->data2;
countertemp=data_val_recv->data3;
countertemp_temp = countertemp;
//tcount++;
//bufsizeinstantt=((ARRAY_LENGTH_DT_TEMP)-tcount);
//printf("\nReceived temperature packet size:  %d      Index:      %li",packetbuf_datalen(),
indexyt);
//printf("\nReceived temperature packets count:      %d",  tempcounttotal);//These
two////////////////////////////////////
//printf("\nNode ID of end device =      %d",  Node_ID);//These
two////////////////////////////////////
//printf("\nCountertemp received =      %d",  countertemp);
//printf("\n-----");

//printf("\n_____
");

```

```
//printf("\n_____")
;
}
}
```

```
/*static void
```

```
broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)
```

```
{
```

```
data_val_recv= (struct transmit_data*)packetbuf_dataptr();
```

```
if(data_val_recv -> type=='L'){
```

```
Light = &data_val_recv->data11;
```

```
Node_ID=data_val_recv->data2;
```

```
counterlight=data_val_recv->data3;
```

```
counterlight_temp = counterlight;
```

```
lightcounttotal++;
```

```
//lightcountwrub++;
```

```
printf("\n\n");
```

```
printf("\n-----");
```

```
printf("\nNode ID of end device =          %d",  Node_ID);
```

```
printf("\nTotal Light packets received at the Coordinator=          %d",  lightcounttotal);
```

```
}
```

```
else if(data_val_recv -> type=='T')
```

```
{
```

```
Temp = &data_val_recv->data11;
```

```
Node_ID=data_val_recv->data2;
```

```
countertemp=data_val_recv->data3;
```

```
countertemp_temp = countertemp;
```



```

tempcounttotal++;
printf("\n\n");
printf("\n-----");
printf("\nNode ID of end device =          %d",  Node_ID);
printf("\nTotal Temperature packets received at the Coordinator=  %d",  tempcounttotal);
}
}*/
/*-----*/
static const struct broadcast_callbacks bc_rx = { broadcast_recv };
static struct broadcast_conn bc;
/*-----*/
void
rt_callback(struct rtimer *t, void *ptr)
{
    leds_off(LEDSD_PERIODIC);
}

/*-----*/
PROCESS_THREAD(cc2538_demo_process, ev, data)
{

    PROCESS_EXITHANDLER(broadcast_close(&bc))

    PROCESS_BEGIN();

    etimercounter = 0;

    broadcast_open(&bc, BROADCAST_CHANNEL, &bc_rx);

```



```

}////4
packetlosslight=((lightcounttotal-lightcounttotal_exit)-ARRAY_LENGTH_RT_LIGHT);
if(packetlosslight<0){//7
printf("\nLight packets lost =                0");           ////////////////////////////////////////////////////
}////5
else{//8
printf("\nLight packets lost =                %d", packetlosslight);
//////////////////////////////////////
}////6
lpcinbuf_temp=lpcinbuf;
lpcinbuf_temp=lpcinbuf_temp*100;
buffer_usage_real_time_light=(lpcinbuf_temp/ARRAY_LENGTH_RT_LIGHT);
buffer_usage_temp_real_time_light=buffer_usage_real_time_light;
buffer_usage_dec_real_time_light=buffer_usage_real_time_light*100;
buffer_usage_decpart_real_time_light=((int)buffer_usage_dec_real_time_light)%100;
buffer_usage_intpart_real_time_light = (int)buffer_usage_temp_real_time_light;
if(buffer_usage_intpart_real_time_light<0){//9
printf("\nLight_buffer_filled =                0.0 percent");
//////////////////////////////////////
printf("\n-----");
printf("\n-----");
}////7
else{//10
if(buffer_usage_intpart_real_time_light>100){//11printf("\n-----
-----");
printf("\nLight_buffer_filled =                100.0 percent");
//////////////////////////////////////
printf("\n-----");
printf("\n-----");
} ////8
else{//12

```

```

        printf("\nLight_buffer_filled =                %d.%d percent",
buffer_usage_intpart_real_time_light,buffer_usage_decpart_real_time_light);
//////////

        printf("\n-----");
        printf("\n-----");
        }///9

}///10
}///11
///12

}

else if(etimer_expired(&et_2))
{
tempcounttotal_exit++;
if(data_val_recv -> type=='T'){
//printf("\nser_countt:                %d    ", ser_countt);
//printf("\nService Rate Buffer Temp array:    %li    ", serviceratebuffertemp[ser_countt]);
printf("\nTotal number of temperature packets serviced =                %d",tempcounttotal_exit);
//////////
serviceratebuffertemp[ser_countt++]=arrtemp[indexyt];
tpcinbuf=tempcounttotal-tempcounttotal_exit;
if(tpcinbuf<0){//3
    printf("\nTemperature packets currently in the buffer =                0");
//////////
}///1
else{//4
    if (tpcinbuf>ARRAY_LENGTH_DT_TEMP){//5
        printf("\nTemperature packets currently in the buffer =                %d",
ARRAY_LENGTH_DT_TEMP);
//////////
}///2
else{//6

```



```

        printf("\n-----");
        //////////////////////////////////

        printf("\n-----");
        //////////////////////////////////

        } ///8
    else{//12

        printf("\nTemperature_buffer_filled =                %d.%d percent",
buffer_usage_intpart_real_time_temp,buffer_usage_decpart_real_time_temp);
        //////////////////////////////////

        printf("\n-----");
        //////////////////////////////////

        printf("\n-----");
        //////////////////////////////////

        }///9

    }

}

}

}

}

PROCESS_END();

}

```

A.3 Codes for both leaf nodes and sink node or coordinator node (Reg. Channel Access Method)

The code for the ‘end device’ node is as below.

```

#include "contiki.h"
#include "cpu.h"
#include "sys/etimer.h"
#include "sys/rtimer.h"
#include "dev/leds.h"

```

```

#include "dev/uart.h"
#include "dev/button-sensor.h"
#include "dev/adc-sensor.h"
#include "dev/watchdog.h"
#include "dev/serial-line.h"
#include "dev/sys-ctrl.h"
#include "net/rime/broadcast.h"
#include <stdio.h>
#include <stdint.h>
#include <math.h>

/*-----*/
#define LOOP_INTERVAL    CLOCK_SECOND
#define LEDS_OFF_HYSTERISIS (RTIMER_SECOND >> 1)
#define LEDS_PERIODIC    LEDS_YELLOW
#define LEDS_BUTTON      LEDS_RED
#define LEDS_SERIAL_IN   LEDS_ORANGE
#define LEDS_REBOOT      LEDS_ALL
#define LEDS_RF_RX       (LEDS_YELLOW | LEDS_ORANGE)
#define BROADCAST_CHANNEL 129
#define N_DECIMAL_POINTS_PRECISION (100)
/*-----*/

static struct etimer et;
static struct rtimer rt;
static uint16_t counter;

//static uint16_t node_ID = 1; //Address 0xCD, 0xCD
//static uint16_t node_ID = 2; //Address 0xCD, 0xCE
//static uint16_t node_ID = 3; //Address 0xCD, 0xCF
//static uint16_t node_ID = 4; //Address 0xCD, 0xDA
//static uint16_t node_ID = 5; //Address 0xCD, 0xDB

```

```
static uint16_t node_ID = 6; //Address 0xCD, 0xDC
//static uint16_t node_ID = 7; //Address 0xCD, 0xDD
//static uint16_t node_ID = 8; //Address 0xCD, 0xDE
//static uint16_t node_ID = 9; //Address 0xCD, 0xDF
//static uint16_t node_ID = 10; //Address 0xCD, 0xEA
//static uint16_t node_ID = 11; //Address 0xCD, 0xEB
//static uint16_t node_ID = 12; //Address 0xCD, 0xEC
//static uint16_t node_ID = 13; //Address 0xCD, 0xED
//static uint16_t node_ID = 14; //Address 0xCD, 0xEE
//static uint16_t node_ID = 15; //Address 0xCD, 0xEF

int Transmit_Flag = 0;
int i = 0;
static uint16_t a[3];
float light_dbl;
int16_t temp;
float temp_dbl;
signed short rssi;
float rssi_dbl;
int lightintpart;
int lightdecpart;
//int lightdecpart_int;
int tempintpart;
int tempdecpart;
//int tempdecpart_int;
int rssiintpart;
int rssiidecpart;
//int rssiidecpart_int;
static uint16_t c[7];
/*-----*/
```



```

PROCESS(cc2538_demo_process, "cc2538 demo process");
AUTOSTART_PROCESSES(&cc2538_demo_process);
/*-----*/
static void
broadcast_rcv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    uint16_t *dataptr_temp;

    dataptr_temp= (uint16_t *)packetbuf_dataptr();

    if(dataptr_temp[0] == node_ID)
    {
        Transmit_Flag=1;
        a[0]=dataptr_temp[0];
        a[1]=dataptr_temp[1];
        a[2]=dataptr_temp[2];
    }
    else
    {
        Transmit_Flag=0;
    }
}
/*-----*/
static const struct broadcast_callbacks bc_rx = { broadcast_rcv };
static struct broadcast_conn bc;
/*-----*/
void
rt_callback(struct rtimer *t, void *ptr)
{
    leds_off(LEDSD_PERIODIC);
}

```

```

}
/*-----*/
PROCESS_THREAD(cc2538_demo_process, ev, data)
{

PROCESS_EXITHANDLER(broadcast_close(&bc))

PROCESS_BEGIN();

counter = 0;
broadcast_open(&bc, BROADCAST_CHANNEL, &bc_rx);

while(1) {

etimer_set(&et, CLOCK_SECOND*0.25);

PROCESS_YIELD();

//if(Transmit_Flag==1)
//{      //leds_toggle(LED_S_RF_RX);
        light_dbl=adc_sensor.value(ADC_SENSOR_ALS);
        if (light_dbl<=4500){
light_dbl= light_dbl*0;
}
if ((light_dbl>=4501) && (light_dbl<=9516)){
light_dbl= ((light_dbl - 4500)/(627/5));
}
if ((light_dbl>=9517) && (light_dbl<=11572)){
light_dbl= ((light_dbl - 5404)/102.8);
}
}
}

```

```

if ((light_dbl>=11573) && (light_dbl<=15744)){
light_dbl= ((light_dbl - 11475)/ (1043/58));
}
if ((light_dbl>=15745) && (light_dbl<=16780)){
light_dbl= ((light_dbl - 14462)/(259/59));
}
if ((light_dbl>=16781) && (light_dbl<=20000)){
light_dbl= ((light_dbl - 16196)/(805/728));
}
if ((light_dbl>=20001) && (light_dbl<=21772)){
light_dbl= ((light_dbl - 19616)*9.28);
}
if ((light_dbl>21773) && (light_dbl<=25116)){
light_dbl= ((light_dbl - 12483)*2.153);
}
if ((light_dbl>25117) && (light_dbl<=25184)){
light_dbl= ((light_dbl - 24730)*70.59);
}

lightintpart = (int)light_dbl;
lightdecpart =
((int)(light_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISI
ON);

temp = adc_sensor.value(ADC_SENSOR_TEMP);
//temp_dbl = (((25 + ((temp >> 4) - 1422) * 10 / 42) - 3)/2);
//temp_dbl=(((temp >> 4) - 1422)*0.2381)+22);

//*****/
/

```

```
//For Node_IDs = 2,5,6 and 10
//temp_dbl((((temp >> 4) - 1422)*0.2381)+22)/2);
//temp_dbl=temp_dbl + 2.56;

//*****/
/

//*****/
/

//For Node_IDs = 8 and 9
//temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
//temp_dbl=temp_dbl-4;

//*****/
/

//*****/
/

//For Node_IDs = 14 and 11
//temp_dbl((((temp >> 4) - 1422)*0.2381)+22);

//*****/
/

//*****/
/

//For Node_IDs = 12, 13 and 3
//temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
//temp_dbl=temp_dbl-1.61;

//*****/
/
```

```
/**
/

//For Node_IDs = 1 and 15
//temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
//temp_dbl=temp_dbl + 1.43;

/**
/

/**
/

//For Node_IDs = 7
//temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
//temp_dbl=temp_dbl - 6.9;

/**
/

/**
/

//For Node_IDs = 4
temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
temp_dbl=temp_dbl - 5.19;

/**
/

/**
//For Node_IDs = 4
temp_dbl((((temp >> 4) - 1422)*0.2381)+22);
```

```

temp_dbl=temp_dbl - 3.25;

/*****/
/

tempintpart = (int)temp_dbl;

tempdecpart =
((int)(temp_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISI
ON);

if (tempdecpart<0) {
tempdecpart = tempdecpart * (-1);
}

rssi=packetbuf_attr(PACKETBUF_ATTR_RSSI);
rssi_dbl=rssi;
rssiintpart = (int)rssi_dbl;

rssiidecpart =
((int)(rssi_dbl*N_DECIMAL_POINTS_PRECISION)%N_DECIMAL_POINTS_PRECISIO
N);

c[0]=node_ID;
c[1]=lightintpart;
c[2]=tempintpart;
c[3]=rssiintpart;
c[4]=lightdecpart;
c[5]=tempdecpart;
c[6]=rssiidecpart;

if(Transmit_Flag==1)
{
printf("This is end device with node_ID=%d transmitting the following values to
the Coordinator.\n", node_ID);

printf("1. Ambient raw light = '%d.%d' lux.\n", lightintpart, lightdecpart);

printf("2. Temperature value = '%d.%d' degree celsius.\n",tempintpart
,tempdecpart);

```

```

    printf("3. rssi      = '%d.%d' dBm.\n", rssiintpart, rssiidecpart);
packetbuf_copyfrom(&c, sizeof(c));
    broadcast_send(&bc);
    Transmit_Flag=0;
    if(a[2]==1)
    {
etimer_set(&et, CLOCK_SECOND*a[1]);
        a[2]=0;
    }
    if(ev == PROCESS_EVENT_TIMER && a[0]!=node_ID)
    {
etimer_set(&et, CLOCK_SECOND*0.25);
    }
}

}

PROCESS_END();
}

```

The code for the ‘coordinator’ or ‘sink’ node is as below.

```

#include "contiki.h"
#include "cpu.h"
#include "sys/etimer.h"
#include "sys/rtimer.h"
#include "dev/leds.h"
#include "dev/uart.h"
#include "dev/button-sensor.h"

```

```

#include "dev/watchdog.h"
#include "dev/serial-line.h"
#include "dev/sys-ctrl.h"
#include "net/rime/broadcast.h"
#include <stdio.h>
#include <stdint.h>
#include <math.h>

/*-----*/
#define LOOP_INTERVAL    CLOCK_SECOND
#define LEDS_OFF_HYSTERISIS (RTIMER_SECOND >> 1)
#define LEDS_PERIODIC    LEDS_YELLOW
#define LEDS_BUTTON      LEDS_RED
#define LEDS_SERIAL_IN   LEDS_ORANGE
#define LEDS_REBOOT      LEDS_ALL
#define LEDS_RF_RX       (LEDS_YELLOW | LEDS_ORANGE)
#define BROADCAST_CHANNEL 129
#define MAX_NODES        15
#define TIME_IN_SEC      7

/*-----*/
/*Coordinator address 0xCD, 0xCC*/
/*-----*/

static struct etimer et;
static struct rtimer rt;
static uint16_t counter;
static uint16_t count_flag;
static uint16_t a[3];
//short signed d[7];
short signed light[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed temperature[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed rssi[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

```



```

short signed lightdec[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed temperaturedec[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short signed rssidec[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

int x;
int y;
/*-----*/
PROCESS(cc2538_demo_process, "cc2538 demo process");
AUTOSTART_PROCESSES(&cc2538_demo_process);
/*-----*/

static void
broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    int16_t *dataptr_temp1;
    //leds_toggle(LED_S_RF_RX);
    dataptr_temp1= (int16_t *)packetbuf_dataptr();
    //if (dataptr_temp1[0]==a[0]){
    //for (x=1;x<=15;x++) {
    //if (a[0]==x) {
    //for (dataptr_temp1[0]=x;dataptr_temp1[0]<=x;dataptr_temp1[0]++){
    x =dataptr_temp1[0];
    light[x] =dataptr_temp1[1];
    temperature[x] =dataptr_temp1[2];
    rssi[x] =dataptr_temp1[3];
    lightdec[x] =dataptr_temp1[4];
    temperaturedec[x] =dataptr_temp1[5];
    rssidec[x] =dataptr_temp1[6];

    if(dataptr_temp1[0] ==2){
    light[dataptr_temp1[0]] =dataptr_temp1[1];
    temperature[dataptr_temp1[0]] =dataptr_temp1[2];

```

```

    rssi[dataptr_temp1[0]] =dataptr_temp1[3];
    lightdec[dataptr_temp1[0]] =dataptr_temp1[4];
    temperaturedec[dataptr_temp1[0]] =dataptr_temp1[5];
    rssidec[dataptr_temp1[0]] =dataptr_temp1[6];
}

```

```

    //if(dataptr_temp1[0]==counter){
//for (counter=1;counter<=15;counter++) {
/*light[dataptr_temp1[0]] =dataptr_temp1[1];
temperature[dataptr_temp1[0]] =dataptr_temp1[2];
rssi[dataptr_temp1[0]] =dataptr_temp1[3];
lightdec[dataptr_temp1[0]] =dataptr_temp1[4];
temperaturedec[dataptr_temp1[0]] =dataptr_temp1[5];
rssidec[dataptr_temp1[0]] =dataptr_temp1[6];*/

dataptr_temp1[0]=0;
dataptr_temp1[1]=0;
dataptr_temp1[2]=0;
dataptr_temp1[3]=0;
dataptr_temp1[4]=0;
dataptr_temp1[5]=0;
dataptr_temp1[6]=0;
//continue;
//}
//}

//}
//if (a[0]==15) {
//a[0] =0;

```

```

//}
//}
//}
}
/*-----*/
static const struct broadcast_callbacks bc_rx = { broadcast_recv };
static struct broadcast_conn bc;
/*-----*/
void
rt_callback(struct rtimer *t, void *ptr)
{
    leds_off(LED_PERIODIC);
}

/*-----*/
PROCESS_THREAD(cc2538_demo_process, ev, data)
{

    PROCESS_EXITHANDLER(broadcast_close(&bc))

    PROCESS_BEGIN();

    counter = 0;
    count_flag=0;
    broadcast_open(&bc, BROADCAST_CHANNEL, &bc_rx);

    while(1) {
        if(count_flag==0)
        {
            etimer_set(&et, CLOCK_SECOND);

```

```

count_flag=1;
}

PROCESS_YIELD();
if(ev == PROCESS_EVENT_TIMER) {
    leds_on(LED_PERIODIC);

    counter++;
    a[0]=counter;
    a[1]=TIME_IN_SEC - 5;
    a[2]=1;
//somewhere is here do the serial output
    etimer_set(&et, CLOCK_SECOND*1);
    packetbuf_copyfrom(&a, sizeof(a));
    broadcast_send(&bc);
    //printf("node_ID=%d\n", a[0]);
    //if (a[0]== 15) {
    //printf("\n\n");

    //}
    if(counter==4)
    {
        counter=(counter % 4);
        //printf("TheString="); ////////////////
printf("|");
for (x=1;x<=4;x++)
{
//print node lines

//printf("|%d,%d,%d,%d|", x, light[x], lightdec[x], temperature[x], temperaturedec[x], rssi[x],
rssi[x]); ////////////////

```

```

//printf("%d,%d,%d,%d", x, light[x], temperature[x], (rssi[x]*(-1))); ////////////////
printf("\n\nEND DEVICE %d: Light=%d lux, Temperature=%d degree celsius(approx.) and
RSSI=%d dBm", x, light[x], temperature[x], rssi[x]);

}

printf("\n");

    etimer_set(&et, CLOCK_SECOND*TIME_IN_SEC);
        }

    }

}

PROCESS_END();
}

```

A.4 Sample Router Fitness Calculation Table for a Given Participant Node towards Electing Replacement Router (To Replace the Departing Router Node)

Parameter	Range	Fiddle Factor (i.e., multiplicand)	Instantaneous (Sensor Value) of the participant node under consideration	Normalized (i.e., equalized value) to be considered for summation
$RSSI_{EDs_AVG}$	0 to $RSSI_{EDs_AVG}$ (max.) (0 to -70 dBm)	$100 \div RSSI_{EDs_AVG}$ (max.) i.e., $[100 \div (-70)]$	($RSSI_{EDs_AVG} =$ say,) -20 dBm	$100 - (\text{Column } 3 \times \text{Column } 4) = 71.42$
$RSSI_{G_ED}$	0 to $RSSI_{G_ED}$ (max.) (0 to -98 dBm)	$100 \div RSSI_{G_ED}$ (max.) i.e., $[100 \div (-98)]$	($RSSI_{G_ED} =$ say,) - 25 dBm	$100 - (\text{Column } 3 \times \text{Column } 4) = 74.48$
BP_{EDs}	0 to BP_{EDs} (max.) (0 to 95%)	$[100 \div BP_{EDs}$ (max.)] i.e., $(100 \div 95)$	($BP_{ED} =$ say,) 95%	$(\text{Column } 3 \times \text{Column } 4) = 100$
Sample router fitness value for the participant router node under consideration				245.9

A.5 Sequence Diagram and Associated Description of All the Messages Transpiring Over the Three Phases of Re-orchestration In Pursuit of Electing the Most Suitable Leaf Node to Take Up the Role of the Replacement Router

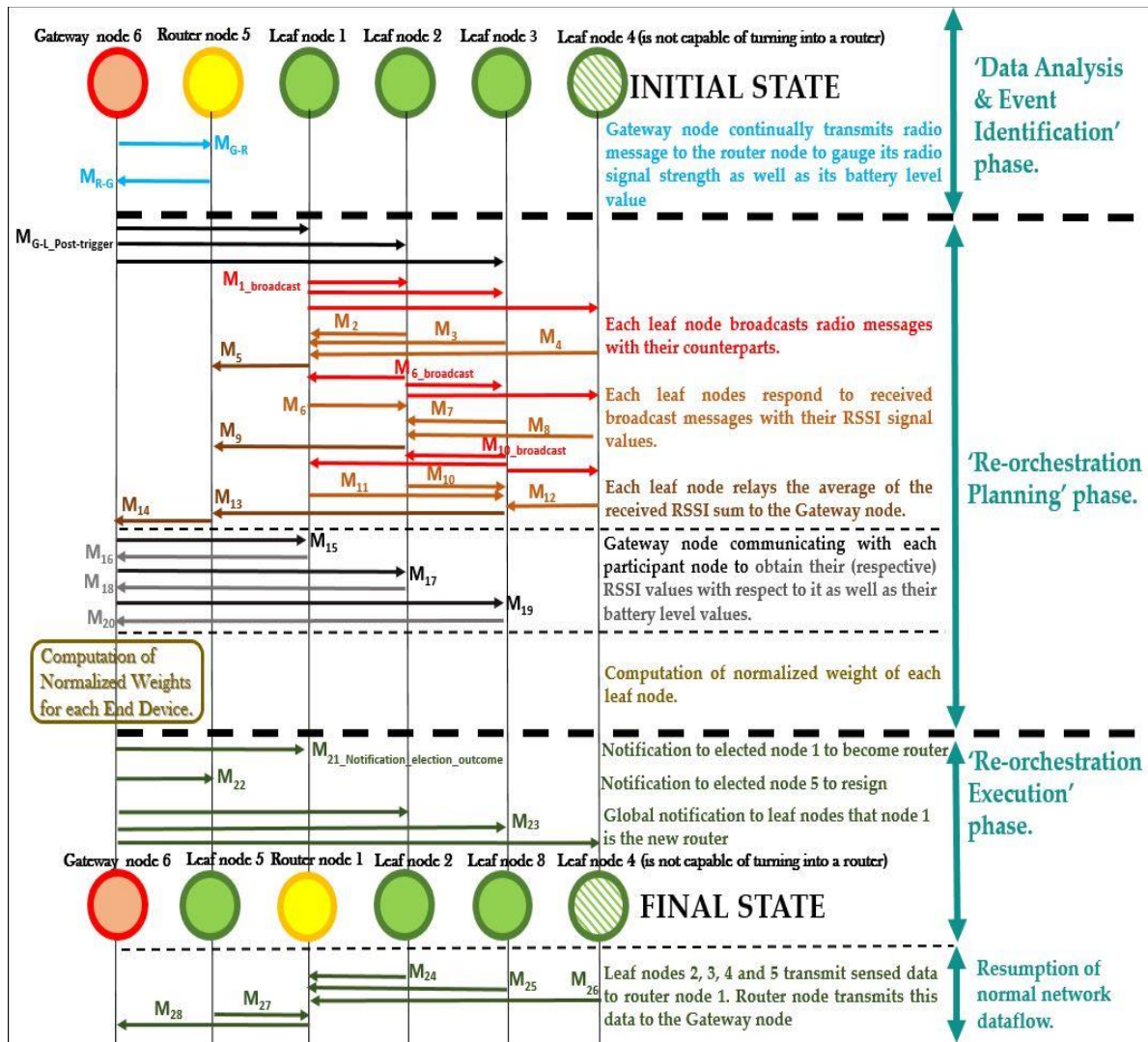


Figure. Sequence diagram showing all the messages transpiring over the three re-orchestration phases in pursuit of electing the most suitable leaf node to take up the role of the replacement router.

During the initial phase of ‘Data Analysis and Event-Identification’, the ‘gateway’ node i.e., the node with node ID 6 continuously transmits the message ‘M_{G-R}’ (at a set rate of transmission) to node ID 5, i.e., the router node, in order to keep a track of its radio signal strength with respect to node ID 5. The router node responds to this message with a message of its own i.e., ‘M_{R-G}’, which also includes its current battery power level. This trend of the RSSI data is stored within the cloud and is continuously monitored by the dedicated ‘knowledge component’ (hosted within the ‘Data and Knowledge’ repository) as part of the monitoring process.

Upon sensing the pattern of increased deviation of the RSSI values between the gateway node and the router node, the ‘Re-orchestration Planning’ phase commences with the gateway node broadcasting a trigger message ‘M_{G-L_Post-trigger}’ to the leaf nodes 1, 2 and 3 (which are capable of assuming the router functionality) to (temporarily) switch to the role of a ‘router’. These newly formed (leaf-turned) ‘router’ nodes (i.e., nodes with node IDs 1, 2 and 3) broadcast messages to each other, including the leaf node 4 (which is not capable of turning into a router), in order to find out their ‘RSSI’ values with respect to each of them. These messages so broadcasted are denoted by messages ‘M_{1_broadcast}’, ‘M_{6_broadcast}’ and ‘M_{10_broadcast}’, transmitted by nodes with node IDs 1, 2 and 3 respectively. These messages are responded to respectively by each of the four-leaf nodes. To elaborate, messages ‘M₂’, ‘M₃’ and ‘M₄’ are transmitted by leaf nodes 2, 3 and 4 respectively, in response to message ‘M_{1_broadcast}’. Similarly, messages ‘M₆’, ‘M₇’ and ‘M₈’ as well as messages ‘M₁₀’, ‘M₁₁’ and ‘M₁₂’ are transmitted by the respective leaf nodes in response to messages ‘M_{6_broadcast}’ and ‘M_{10_broadcast}’ respectively. All the participant nodes (i.e., nodes 1, 2 and 3) relay the average value of RSSI signal values received by them over to node 5, i.e., the router node, as denoted by messages ‘M₅’, ‘M₉’ and ‘M₁₃’ respectively. This value is relayed by the router node over to node ID 6 (i.e., the Gateway node) through message ‘M₁₄’. Through transmission of the messages ‘M₁₅’, ‘M₁₇’ and ‘M₁₉’ to the nodes three participant nodes, the Gateway node then seeks to find out its respective RSSI as well as battery power level values with each of them. In response, each of the three leaf nodes transmit messages ‘M₁₆’, ‘M₁₈’ and ‘M₂₀’ respectively (consisting of their battery level values, as well) to the node 6 (gateway node).

The final phase of ‘Re-orchestration Execution’ involves implementation of the outcomes derived within the preceding phase of ‘Re-orchestration Planning’. Here, the following messages get exchanged amongst the constituent nodes in a sequential manner (as depicted in the above figure).

- The gateway node transmits message ‘M_{21_Notification_election_outcome}’ to the participant leaf node with the highest normalized weight value (i.e., to the leaf node 1, in this case) ‘directing’ it to take up the ‘role’ of the ‘replacement router’.
- The outgoing router then receives message ‘M₂₂’ from the gateway node directing it to stop acting as a router and assume the function of a leaf node.
- By means of broadcasting ‘M₂₃’ message, gateway node 6 informs all the constituent nodes about node 1 being the new replacement router.
- Upon switching over to the role of replacement router, node 1 (traverses to a suitable location to be within the range of the gateway as well as the constituent leaf nodes) and thereby facilitates for the resumption of the normal flow of data within the network wherein it relays the ‘sensed data’ obtained from the all the ‘leaf’ nodes (as indicated by the messages ‘M₂₄’, ‘M₂₅’, ‘M₂₆’ and ‘M₂₇’) and relays it over to the ‘Gateway’ node (as indicated by message ‘M₂₈’).

A.6 Sample Router Fitness Calculation Table for a Given Participant Node towards Electing Replacement Router (To Replace the Dying Router Node)

Parameter	Range	Fiddle Factor (i.e., multiplicand)	Instantaneous (Sensor Value) of the participant node under consideration	Normalized (i.e., equalized value) to be considered for summation
$RSSI_{RCL-G}$	0 to $RSSI_{RCL-G}$ (max.) (0 to -70 dBm)	$100 \div RSSI_{RCL-G}$ (max.) i.e., $[100 \div (-98)]$	($RSSI_{RCL-G} =$ say,) -30 dBm	$100 - (\text{Column 3} \times \text{Column 4}) = 69.38$
$RSSI_{RCL-L}$	0 to $RSSI_{RCL-L}$ (max.) (0 to -98 dBm)	$100 \div RSSI_{RCL-L}$ (max.) i.e., $[100 \div (-98)]$	($RSSI_{RCL-L} =$ say,) -35 dBm	$100 - (\text{Column 3} \times \text{Column 4}) = 64.28$
C_{CL-RCL}	0 to C_{CL-RCL} (max.) (0 to 4)	$100 \times \{1 - [C_{CL-RCL} / C_{CL-RCL} \text{ (max.)}]\}$	($C_{CL-RCL} =$ say,) 4	0
B_{RCL}	0 to B_{RCL} (max.) (0 to 95%)	$[100 \div B_{RCL}$ (max.)] i.e., $(100 \div 95)$	($B_{RCL} =$ say,) 92%	(Column 3 \times Column 4) = 96.84
Sample router fitness value for the participant router node under consideration				230.5