TUM School of Computation, Information and Technology
Technische Universität München

Clustering-based Solutions for Energy Efficiency, Adaptability and Resilience in IoT Networks

Nitin Shivaraman

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzende:
Prof. Dr. Antonia Wachter-Zeh

Prüfende der Dissertation:
1. Prof. Dr.-Ing. Sebastian Steinhorst
2. Prof. Dr. Arvind Easwaran,
   Nanyang Technological University, Singapur

Die Dissertation wurde am 11.04.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 17.09.2022 angenommen.

Abstract

The Internet-of-Things (IoT) paradigm comprises devices ranging from simple sensors to servers that collect, process, and communicate information between each other. Among them, resource-constrained devices (constrained in processor, memory and battery) are deployed in most IoT applications due to their low cost and high availability. Communication between these devices is the highest energy consuming operation and hence, needs to be efficient to sustain their operations in a network over long periods. Additionally, devices may join, leave or move across the network in an ad-hoc manner. IoT devices must be adaptable to these dynamic changes and maintain a stable load on the network. Owing to their low-cost nature and susceptibility to damage, tampering or battery exhaustion, some devices may become faulty leading to unexpected behavior. IoT networks need to remain functional and resilient even with the presence of such faulty devices. Hence, this thesis identifies three major requirements for IoT networks: energy efficiency, adaptability, and fault resilience.

It is very challenging to collectively address all three requirements as they have an orthogonal function to each other. *E.g.* fault-resilient solutions use complex computations for verification of devices and data that inadvertently reduce the energy efficiency in a resource-constrained environment. Existing solutions achieve one or at most two of the requirements collectively, with no solution incorporating all three requirements. Given the large-scale deployment of IoT devices and their price-sensitive nature, it is imperative to design algorithmic solutions to meet the three requirements as this thesis proposes. In this regard, we propose several communication algorithms to cover the three requirements in this thesis.

In the first part of the thesis, we introduce a novel communication algorithm DeCoRIC, that dynamically groups asynchronous devices into clusters and elects a representative among them. DeCoRIC strikes the right balance in the communication load to reduce energy consumption while maintaining connectivity across different devices in the network. Further, unlike most existing clustering solutions which are susceptible to faults and network partitions, we also achieve adaptability and resilience by utilizing message *gossips* to inform device status among neighboring devices, allowing for alternate connecting paths in the event of a faulty device. We evaluate DeCoRIC on the Contiki simulation platform and compare it against state-of-the-art clustering solutions. We find that DeCoRIC improves the power efficiency of the network by at least 70% and extends its lifetime by at least 42% over existing solutions while making the network adaptable and resilient to faults.

Further, in the second part, we showcase the practical applicability of DeCoRIC with a use-case of a clustered network of energy resources and propose a charge scheduling algorithm to improve their utilization. In particular, we investigate resource allocation and scheduling of devices in an electric grid network to maximize energy utilization.

Through experiments, we found that the mobility of devices allows them to be adaptable by moving to different clusters (geographical location) for energy consumption if the current cluster is overloaded. Hence, we devise an online load scheduling algorithm that exploits different charging modes and mobility of devices to maximize the devices that meet their energy demands. Our solution creates a feasible schedule for the devices in a fraction of time (in the order of minutes) compared to conventional optimization solvers (in the order of hours). We also tested our solution on a real-world electric vehicle dataset and found a 57% improvement in the achieved utility compared to other scheduling solutions such as earliest deadline first or highest-energy demand scheduling.

To further improve the energy efficiency and tolerance of the IoT networks, the last part of the thesis builds on DeCoRIC to synchronize the devices and create a timed communication. While the literature on time synchronization addresses energy efficiency and adaptability partially, none of the solutions provide resilience to faults. Therefore, we design a novel time synchronization solution *C-sync*, that is fault-resilient and adaptable while maintaining microsecond (µs) accuracy. In addition, C-sync uses a byzantine consensus mechanism to identify anomalies in the device communication and ensures correct information is propagated across the network. Further, we introduce a concept of *local centers* to limit the maximum number of hops a device is located from its time source making the network adaptable. C-sync is tested on a hardware testbed and shown to consume at least 51% lower power than existing state-of-the-art resilient synchronization solutions. The resilience of C-sync to byzantine faults is also demonstrated by a quick recovery of correct time information when faulty information was introduced.

The algorithms presented in this thesis collectively achieve an energy-efficient, adaptable and fault-resilient IoT network organically to solve the research gaps despite the orthogonal requirements. This thesis provides the first step in advancing IoT application domains limited by these missing requirements. Applications with a large number of IoT devices such as smart factories and smart cities benefit from the presented adaptable clustered network solution with its real-time monitoring and control. Resources are better utilized and congestion is reduced by the underlying consensus mechanisms while allowing cost-effective equipment maintenance using the fault-resilience capabilities of our solution. Implementations on real-world data and testbeds used for experiments elucidate the practical viability of the solutions for IoT networks.

Zusammenfassung

Das Internet-der-Dinge-Paradigma (IoT) umfasst Geräte, die von einfachen Sensoren bis hin zu Servern reichen, die Informationen sammeln, verarbeiten und miteinander kommunizieren. In den meisten IoT-Anwendungen werden ressourcenbeschränkte Geräte (mit eingeschränktem Prozessor, Speicher und Akku) aufgrund ihrer geringen Kosten und hohen Verfügbarkeit eingesetzt. Die Kommunikation zwischen diesen Geräten muss effizient sein, um ihren Betrieb in einem Netzwerk über lange Zeiträume aufrechtzuerhalten, da Funkverbindungen von allen Ressourcen die meiste Energie verbrauchen. Außerdem können sich Geräte ad hoc mit dem Netz verbinden, es verlassen oder sich innerhalb des Netzes bewegen. IoT-Geräte müssen an diese dynamischen Veränderungen angepasst werden können und eine stabile Belastung des Netzes gewährleisten. Aufgrund ihrer geringen Kosten und ihrer Anfälligkeit für Beschädigungen, Manipulationen oder erschöpfte Batterien können einige Geräte fehlerhaft werden, was zu unerwartetem Verhalten führt. IoT-Netzwerke müssen auch bei Vorhandensein solcher fehlerhaften Geräte funktionsfähig und widerstandsfähig bleiben. Daher werden in dieser Arbeit drei Hauptanforderungen an IoT-Netzwerke identifiziert: Energieeffizienz, Anpassungsfähigkeit und Fehlerresistenz.

Es ist eine große Herausforderung, alle drei Anforderungen gleichzeitig zu erfüllen, da sie orthogonal zueinander funktionieren. *Z.B.* verwenden fehlerresistente Lösungen komplexe Berechnungen zur Überprüfung von Geräten und Daten, die unbeabsichtigt die Energieeffizienz in einer ressourcenbeschränkten Umgebung verringern. Bestehende Lösungen erfüllen eine oder höchstens zwei der Anforderungen gleichzeitig, wobei keine Lösung alle drei Anforderungen erfüllt. Angesichts des großflächigen Einsatzes von IoT-Geräten und ihrer preissensiblen Natur ist es zwingend erforderlich, algorithmische Lösungen zu entwickeln, die die drei Anforderungen erfüllen, wie sie in dieser Arbeit vorgeschlagen werden. In diesem Zusammenhang schlagen wir in dieser Arbeit mehrere Algorithmen zur Abdeckung der drei Anforderungen vor.

Im ersten Teil der Arbeit stellen wir einen neuartigen Kommunikationsalgorithmus DeCoRIC vor, der asynchrone Geräte dynamisch in Clustern gruppiert und einen Repräsentanten unter ihnen wählt. DeCoRIC sorgt für das richtige Gleichgewicht in der Kommunikationslast, um den Energieverbrauch zu senken und gleichzeitig die Konnektivität zwischen den verschiedenen Geräten im Netzwerk aufrechtzuerhalten. Im Gegensatz zu den meisten existierenden Clustering-Lösungen, die anfällig für Fehler und Netzwerkpartitionen sind, erreichen wir außerdem Anpassungsfähigkeit und Ausfallsicherheit, indem wir Nachrichten *gossips* verwenden, um den Gerätestatus benachbarter Geräte mitzuteilen, was im Falle eines fehlerhaften Geräts alternative Verbindungswege ermöglicht. Wir evaluieren DeCoRIC auf der Simulationsplattform Contiki und vergleichen es mit modernen Clustering-Lösungen. Wir stellen fest, dass DeCoRIC die Energieeffizienz des Netzwerks um mindestens 70% verbessert und seine Lebensdauer

um mindestens 42% im Vergleich zu bestehenden Lösungen verlängert, während das Netzwerk anpassungsfähig und widerstandsfähig gegenüber Fehlern ist.

Im zweiten Teil zeigen wir die praktische Anwendbarkeit von DeCoRIC anhand eines Anwendungsfalls eines gebündelten Netzwerks von Energieressourcen und schlagen einen Algorithmus zur Gebührenplanung vor, um deren Nutzung zu verbessern. Insbesondere untersuchen wir die Ressourcenzuweisung und -planung von Geräten in einem Stromnetz, um die Energienutzung zu maximieren. In Experimenten haben wir herausgefunden, dass die Mobilität der Geräte es ihnen ermöglicht, sich anzupassen, indem sie in andere Cluster (geografische Standorte) wechseln, um Energie zu verbrauchen, wenn der aktuelle Cluster überlastet ist. Daher entwickeln wir einen Online-Lastplanungsalgorithmus, der verschiedene Lademodi und die Mobilität der Geräte ausnutzt, um die Geräte zu maximieren, die ihre Energieanforderungen erfüllen. Unsere Lösung erstellt einen praktikablen Zeitplan für die Geräte in einem Bruchteil der Zeit (in der Größenordnung von Minuten) im Vergleich zu herkömmlichen Optimierungslösungen (in der Größenordnung von Stunden). Wir haben unsere Lösung auch an einem realen Elektrofahrzeugdatensatz getestet und eine 57%ige Verbesserung des erzielten Nutzens im Vergleich zu Standardplanungslösungen festgestellt.

Um die Energieeffizienz und Toleranz der IoT-Netzwerke weiter zu verbessern, baut der letzte Teil der Arbeit auf DeCoRIC auf, um die Geräte zu synchronisieren und eine zeitgesteuerte Kommunikation aufzusetzen. Während die Literatur zur Zeitsynchronisation Energieeffizienz und Anpassungsfähigkeit teilweise anspricht, bietet keine der Lösungen Resilienz gegenüber Fehlern. Daher entwerfen wir eine neuartige Zeitsynchronisationslösung *C-sync*, die fehlerresistent und anpassungsfähig ist und gleichzeitig eine Genauigkeit von Mikrosekunden (µs) beibehält. C-sync verwendet einen byzantinischen Konsensmechanismus, um Anomalien in der Gerätekommunikation zu erkennen und sicherzustellen, dass korrekte Informationen über das Netzwerk verbreitet werden. Darüber hinaus führen wir ein Konzept der *local centers* ein, um die maximale Anzahl von Hops zu begrenzen, die ein Gerät von seiner Zeitquelle entfernt ist, was das Netzwerk anpassungsfähig macht. C-sync wurde auf einem Hardware-Testbed getestet und es wurde gezeigt, dass es mindestens 51% weniger Energie verbraucht als bestehende, dem Stand der Technik entsprechende Synchronisationslösungen. Die Widerstandsfähigkeit von C-sync gegenüber byzantinischen Fehlern wird auch durch eine schnelle Wiederherstellung der korrekten Zeitinformationen demonstriert, wenn fehlerhafte Informationen eingeführt wurden.

Die in dieser Arbeit vorgestellten Algorithmen erreichen gemeinsam ein energieeffizientes, anpassungsfähiges und fehlerresistentes IoT-Netzwerk, um die Forschungslücken trotz der orthogonalen Anforderungen organisch zu lösen. Diese Arbeit stellt den ersten Schritt dar, um IoT-Anwendungsbereiche voranzubringen, die durch diese fehlenden Anforderungen eingeschränkt sind. Anwendungen mit einer großen Anzahl von IoT-Geräten, wie z. B. intelligente Fabriken und intelligente Städte, profitieren von der vorgestellten anpassungsfähigen geclusterten Netzwerklösung mit ihrer Echtzeitüberwachung und -steuerung. Die zugrundeliegenden Konsensmechanismen sorgen für eine bessere Ressourcennutzung und eine geringere Überlastung, während die Fehlerresistenz unserer Lösung eine kostengünstige Wartung der Geräte ermöglicht. Im-

plementierungen auf realen Daten und Testbeds, die für Experimente verwendet werden, verdeutlichen die Praxistauglichkeit der Lösungen für IoT-Netzwerke.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors Arvind Easwaran and Sebastian Steinhorst. They provided me with this valuable opportunity to pursue a Ph.D. in the best possible setup. Arvind helped improve my research by asking hard questions on the hypothesis and has always made himself available during my deadlines for multiple meetings a week. Sebastian has kept me motivated on multiple occasions and has an incredible eye for reviews in spotting minor errors which were critical yet overlooked. Providing me with an external reviewer's perspective of my research, both supervisors have inculcated me with the habit of producing very high-quality research that is transparent, understandable and reproducible.

I would also like to thank Shreejith Shanker and Saravanan Ramanathan for their valuable feedback and help in shaping my research. Shreejith has been indispensable for guiding me through the early years of my Ph.D. and establishing the foundation needed for this thesis. Saravanan's vigor in research pushed me to take on more challenging topics and ideas.

I would like to thank my colleagues in ESTL, TUMCREATE including Sebastian, Kai, Alex, Marc, Srikkanth, Andrej, Arif and Tobias for the amazing company throughout the past three years. I would also like to thank my NTU colleagues including Ankita, Sriram, Vijay, Zhonglin, Daniel and my TUM colleagues Emanuel, Laurin and Hamad. It was a wonderful experience collaborating with Emanuel on our joint project. Discussions with Ankita resonated with some of my own research problems. I would also like to thank Prof. Jialin Li and Prof. Etienne Borde for providing me an opportunity to collaborate on different occasions.

It is important to mention my friends Ankit, Indermohan, Kashyap, Shiva and Neelesh for their encouragement through this period. Chaitanya, Prabhuraj and Vivek helped create a conducive environment at home.

My doctoral journey has been sane and less stressful thanks to my wife Trendy for her unwavering love and support. She also helped me refine my writing skills on multiple occasions. Trendy's parents Kay Tong and Han Soo have treated me like their son and backed me during difficult times.

A special thanks to Arvind, Abhiruchi, Murali, Priya and Moksh for making me feel at home and treating me to delicious food. Lengthy discussions with Moksh pushed me towards top conferences and to kept my research quality high.

Finally, my heartfelt thanks to my parents who showered me with unconditional love and stayed with me through thick and thin. They have encouraged me at every step and provided me with valuable motivation to push myself to be better. They kept me away from financial pressures even during difficult times to prevent me from getting distracted from research. It is not an exaggeration to say that I could not have made it through this difficult and long journey without my parents and my wife.

Publications

First-authored publications

Nitin Shivaraman, Patrick Schuster, Saravanan Ramanathan, Arvind Easwaran, Sebastian Steinhorst, "Cluster-based Network Time Synchronization for Resilience with Energy Efficiency," 2021 IEEE Real-Time Systems Symposium (RTSS), 2021.

Nitin Shivaraman, Jakob Fittler, Saravanan Ramanathan, Arvind Easwaran, Sebastian Steinhorst, "A novel load distribution strategy for aggregators using IoT-enabled mobile devices", 2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), 2021.

Nitin Shivaraman, Saravanan Ramanathan, Shreejith Shanker, Arvind Easwaran, Sebastian Steinhorst, "DeCoRIC: Decentralized Connected Resilient IoT Clustering," 2020 29th International Conference on Computer Communications and Networks (ICCCN), 2020.

Nitin Shivaraman, Patrick Schuster, Saravanan Ramanathan, Arvind Easwaran, Sebastian Steinhorst, "Poster Abstract: C-Sync: The Resilient Time Synchronization Protocol," 2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 2020.

Nitin Shivaraman, Jakob Fittler, Saravanan Ramanathan, Arvind Easwaran, Sebastian Steinhorst, "WiP Abstract: Mobility-based Load Balancing for IoT-enabled Devices in Smart Grids," 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS), 2020.

Nitin Shivaraman, Seima Saki, Zhiwei Liu, Saravanan Ramanathan, Arvind Easwaran, Sebastian Steinhorst, "Real-Time Energy Monitoring in IoT-enabled Mobile Devices," 2020 Design, Automation and Test in Europe Conference Exhibition (DATE), 2020.

Nitin Shivaraman, Arvind Easwaran, Sebastian Steinhorst, "Efficient decentralized active balancing strategy for smart battery cells," Design, Automation and Test in Europe Conference Exhibition (DATE), 2017, 2017.

Co-authored publications

Saravanan Ramanathan, Nitin Shivaraman, Seima Suryasekaran, Arvind Easwaran, Etienne Borde and Sebastian Steinhorst, "A survey on time-sensitive resource allocation in the cloud continuum", it - Information Technology, 2020.

Emanuel Regnath, Nitin Shivaraman, Shanker Shreejith, Arvind Easwaran, Sebastian Steinhorst, "Blockchain, what time is it? Trustless Datetime Synchronization for IoT," 2020 International Conference on Omni-layer Intelligent Systems (COINS), 2020.

Contents

Abstract iii

Zusammenfassung v

Acknowledgements ix

Publications xi

Contents xiii

List of Figures xvii

List of Tables xix

Acronyms xxi

1 Introduction 1
- 1.1 Major requirements in designing an IoT solution . . . 1
- 1.2 Challenges to combining the requirements . . . 3
- 1.3 Existing solutions address a subset of the major requirements . . . 4
- 1.4 Research Contributions . . . 6
- 1.5 Organization . . . 7

2 Background 9
- 2.1 Communication topologies . . . 9
  - 2.1.1 Centralized network . . . 9
  - 2.1.2 Decentralized network . . . 9
  - 2.1.3 Clustered network . . . 10
- 2.2 Communication primitives . . . 11
  - 2.2.1 Frame Structure . . . 11
- 2.3 Network Model . . . 12
  - 2.3.1 Model assumptions . . . 13
  - 2.3.2 Fault model . . . 14
- 2.4 Simulation Platforms . . . 15
  - 2.4.1 Contiki Operating System . . . 15

3 DeCoRIC: Energy-efficient, connected and resilient clustering 19
- 3.1 Related work . . . 21

3.2 DeCoRIC strategy . . . 24
3.2.1 Radio primitives . . . 24
3.2.1.1 CSMA-CA . . . 24
3.2.1.2 Radio duty cycling . . . 25
3.2.2 Clustering state machine . . . 25
3.3 Analysis & evaluation . . . 33
3.3.1 Power consumption . . . 35
3.3.2 Connectivity . . . 39
3.3.3 Evaluation of resilience . . . 41
3.4 Summary . . . 43

4 DeCoRIC use-case: Load Balancing for Mobile devices 45
4.1 Device mobility evaluation . . . 47
4.2 Related Work . . . 50
4.3 System model . . . 52
4.3.1 Use-case-specific assumptions . . . 52
4.3.2 Network-level parameters . . . 54
4.3.3 Device-level parameters . . . 54
4.3.4 Utility function . . . 55
4.3.5 Objective function and constraints . . . 57
4.4 Low-complexity heuristic solution . . . 60
4.5 Experiments . . . 63
4.5.1 Optimization platforms . . . 63
4.5.2 Heuristic performance . . . 64
4.5.3 Comparison with solver . . . 66
4.5.4 Evaluation on a real-world dataset . . . 67
4.6 Summary . . . 67

5 C-sync: Energy-efficient and resilient time synchronization using clustered networks 69
5.1 Related work . . . 72
5.2 C-sync protocol . . . 76
5.2.1 Clustering phase . . . 77
5.2.2 Consensus phase . . . 81
5.3 Resilience in C-sync . . . 86
5.3.1 Network assumptions to achieve byzantine fault resilience . . . 87
5.3.2 Fault detection and correction . . . 88
5.4 Experiments and results . . . 90
5.4.1 Experimental setup . . . 91
5.4.2 Fault detection and correction . . . 93
5.4.3 Energy efficiency for fault-tolerance . . . 95
5.4.4 Synchronization error to local center . . . 98
5.5 Summary . . . 99

6 Conclusion & Future Work **101**
6.1 Thesis Summary 101
6.2 Future work 103
6.2.1 DeCoRIC: 103
6.2.2 Use-case of DeCoRIC - Load balancing for mobile devices: 104
6.2.3 C-sync: 104

Bibliography **107**

List of Figures

1.1 Major requirements for an IoT network. Existing solutions meet at most two requirements indicated by shaded area while no solution meets all the requirements. . . . 3

2.1 Common topologies used in IoT networks. . . . 10
2.2 A clustered network combining the properties of centralized and decentralized networks. . . . 11
2.3 The frame structure of the IEEE 802.15.4 protocol at the Physical and medium access control layers. . . . 12
2.4 Different types of Graphs based on the direction of communication links among the nodes. . . . 13

3.1 Clustering operation: starting with any topology (a), the nodes align themselves into clusters with an elected CH (b). With DeCoRIC, the clustering dynamically adapts to changes in topology to ensure connectivity among all nodes (c). . . . 20
3.2 DeCoRIC states and transition conditions. . . . 26
3.3 DeCoRIC message format. . . . 26
3.4 DeCoRIC on an example network (not drawn to scale). . . . 27
3.5 Number of CH nodes and average power consumed by the nodes running DeCoRIC over different RSSI thresholds to form clusters. . . . 36
3.6 Average Power Consumption vs Time of first node death in the network for DeCoRIC compared against LEACH and BEEM. . . . 37
3.7 Battery drain of the network of DeCoRIC, LEACH and BEEM. The gray area indicates the variation between the minimum and maximum number of alive nodes with the solid line representing the average number of alive nodes. . . . 38
3.8 Power consumed and time taken for the network clustering normalized by connectivity among all the nodes in the network. . . . 40
3.9 Time taken for the network to re-cluster and stabilize in case of a node death. . . . 43

4.1 The grid aggregator and the devices (including mobile devices formed into clusters) . . . 46
4.2 Device movement between two clusters and current consumption from different aggregators. . . . 48
4.3 Device registration and handshake messages for mobility between two clusters. . . . 49

4.4 Device transition of current consumption and reporting after moving from cluster 1 to 2. 50
4.5 The difference of utility loss achieved with and without mobility 65
4.6 Comparison of solver performance with the heuristic for the same runtime. 66

5.1 Time Synchronization in C-sync using a clustered architecture for power efficiency and fault resilience. 70
5.2 The two phases of the C-sync protocol represented as a state machine. . . 76
5.3 The process of finding LCs from the edge CHs and their associated nodes which receive time information. 83
5.4 Example of consensus states with 10 slots. 84
5.5 Software architecture of Contiki OS integrated with C-sync. 91
5.6 DCO time drift compensation using the stable low-frequency crystal oscillator. 93
5.7 The synchronization error of different nodes within a cluster having a faulty CH. The error remains within one logical time slot without any impact on network synchronization. 94
5.8 Scatter plot of average neighbor synchronization error and the average power consumption for each node over different topologies that are densely distributed, sparsely distributed and using the entire testbed compared for C-sync and GTSP. 96
5.9 Chain of Cluster Heads and Cluster Bridge ignoring the Common nodes to demonstrate a multi-hop network. 98
5.10 Synchronization error is bounded by the established the maximum number of hops from the local center. 99

List of Tables

3.1 Comparison of notable works in Literature. 23
3.2 Parameters used in our experimental setup for evaluating DeCoRIC against LEACH and BEEM. 35
3.3 Best and worst-case reaction time at each node with DeCoRIC in case of network changes. 42

4.1 Notations and associated description used for modeling. 53
4.2 Parameters used in our experimental setup for evaluating the proposed algorithm. 64
4.3 Number of samples generated for the solver. 67
4.4 Utility loss for EV dataset with different strategies. 67

5.1 Comparison of prominent synchronization solutions in the literature. . . . 72
5.2 Results of average neighbor synchronization error and power consumption measured (with associated standard deviation) over 30 minutes across different topologies. 97

Acronyms

6LowPAN	IPv6 over Low-Power Wireless Personal Area Networks.
AES	Advanced Encryption Standard.
BE	Back-off Exponent.
BFT	Byzantine Fault Tolerance.
BPSK	Binary Phase Shift Keying.
CB	Cluster Bridge.
CBH	Cluster Bridge Head.
CCA	Clear Channel Assessment.
CH	Cluster Head.
CM	Common Node.
COTS	Commercial Off-The-Shelf.
CRC	Cyclic Redundancy Check.
CSMA-CA	Carrier Sense Multiple Access - Collision Avoidance.
CSMA-CD	Carrier Sense Multiple Access - Collision Detection.
CTS	Clear to Send.
DCO	Digitally Controlled Oscillator.
DER	Distributed Energy Resource.
DSSS	Direct-Sequence Spread Spectrum.
EV	Electric Vehicle.
FCS	Frame Check Sequence.
FSM	Finite State Machine.
GCD	Greatest Common Divisor.
HVAC	Heating Ventilation Air Conditioning.
IoT	Internet of Things.
ITU	International Telecommunications Union.
LC	Local Center.

LCM	Lowest Common Multiple.
MAC	Medium Access Control.
MINLP	Mixed-Integer Non-Linear Programming.
OSI	Open Systems Interconnection.
QPSK	Quadrature Phase Shift Keying.
RDC	Radio Duty Cycling.
RSSI	Received Signal Strength Indicator.
RTS	Request to Send.
SNR	Signal to Noise Ratio.
SoC	State-of-Charge.
TDMA	Time Division Multiple Access.
WirelessHART	Wireless Highway Addressable Remote Transducer Protocol.
WSN	Wireless Sensor Networks.

1 Introduction

The Internet-of-Things (IoT) paradigm has been rapidly adopted in a plethora of applications, including but not limited to smart homes, Industry 4.0 and smart cities amongst others. According to the definition by the International Telecommunication Union (ITU), IoT is defined as any system comprising a network of devices, where each device (also referred to as *thing/node/end-device*) has at least one transducer (sensor or actuator) for interacting directly with the physical world, has at least one network interface connecting the device to the internet/local network directly or indirectly through a wired/wireless medium and can function independently with/without a processor [1]. However, it is not mandatory for the devices to be connected to the public internet. Combining the sensors and actuators of the physical system with the digital capabilities of processors establishes the *cyber-physical* scope of IoT.

1.1 Major requirements in designing an IoT solution

Commercial off-the-shelf (COTS) cyber-physical devices interacting over various wireless communication standards such as Wi-Fi, Zigbee, etc., form the bulk of apparatus in most IoT applications [2, 3]. These IoT devices are primarily low-cost and battery-operated with limited computational (processor) and communication (radio) resources [4]. A common challenge in devising *communication strategies/algorithms* for such applications is the high proportion of energy consumption arising from data transfer [5]. Empirical results have shown a single data communication operation from the radio translates to hundreds of computations on the processor [6]. Data communication *wakes up* the device from its sleep state coupled with processing operations to handle the data. Due to its high energy requirement, efficient usage of radio by minimizing communication is critical to maintaining a long operational lifetime, especially for applications that require frequent communication among devices. For example, environmental sensing in harsh weather locations or industrial manufacturing plants require constant monitoring of the surroundings and communicating the sensed data frequently between devices. Finding the right balance in the periodicity of communication enables battery conservation and extends the lifetime of the devices.

Though energy efficiency is addressed in existing literature [5, 7], IoT applications struggle to maintain consistent operations in the face of complex and dynamic network topologies due to frequent device movement, additions or removals. In essence, the devices may join or leave the network at any time in an ad-hoc manner which may overload or underload other devices, thus disrupting stable operations in the network. For instance, sensors in application domains such as environmental sensing, industrial plants, etc., may be moved to other regions or damaged by natural disasters, moving parts

or harsh operating conditions. Due to the difficulty in replacing devices under these challenging conditions, new devices are added to the network regularly; the additional devices must maintain a steady communication load to ensure stability in network operations. To deal with this dynamism in the network, existing solutions include having a central device powerful enough to handle overloads through data aggregation [8], a hierarchy of devices to disseminate and process data [9], etc. However, adaptability of IoT networks to device dynamism in addition to being energy-efficient adds a new dimension to the requirements.

Furthermore, IoT networks must sustain their functionality even in the presence of faulty end-devices. A fault indicates a device behavior that does not adhere to its expected functionality. A faulty device could impact the network operation in several ways: *silent* devices can partition the network resulting in connectivity issues, whereas devices transmitting erroneous information may result in unexpected behaviors. Hence, it is essential for IoT networks to maintain stable operations and be *resilient* to such faulty devices. Referring to our examples of environmental sensing and industrial manufacturing plants, IoT networks must dynamically adjust themselves to maintain their functionality regardless of faulty devices to ensure critical events such as an earthquake or a failed robotic arm are reported quickly and correctly. By general scientific notion, resilience to faulty devices can be achieved through redundancy of end-devices, information validation during communication, etc. [10, 11, 12]. Nonetheless, integrating the requirements of either energy efficiency or adaptability with fault resilience poses a non-trivial challenge.

While solutions such as increasing redundancy and battery capacity or the use of resource-rich devices offer fault protection mechanisms, these methods are price-sensitive and increase the complexity of network operations. These expensive devices mandate regular maintenance that would further add to costs. Hence, in order to maintain a long lifetime, adapt to changing network topology and ensure resilience to faulty devices, the design of IoT applications has three major requirements: 1) Energy efficiency. 2) Adaptability 3) Resilience to device faults. These three requirements are essential across various application domains with large-scale IoT networks such as environmental monitoring (*e.g.* home, industry, weather), agriculture, etc.

A practical approach to avoid high costs or hardware redundancy is to dynamically adapt any new network changes through an algorithmic solution to integrate the three requirements. Device dynamism can be handled in an energy-efficient IoT network by designing a communication strategy to dynamically adapt to the changes. Similarly, neighboring end-devices must communicate the presence of faulty devices among each other to prevent the faults from propagating through the network. In summary, a communication algorithm that incorporates all the three requirements of energy efficiency, resilience and adaptability by design is pivotal to ensuring a long and stable operation of IoT networks.

Figure 1.1 shows the major requirements of IoT networks that are the focus of this thesis. Very few solutions in the literature have simultaneously addressed both the requirements of energy efficiency and adaptability represented by the grey shaded area between the orange and green circles in Figure 1.1 [13, 14]. Similarly, the shaded areas

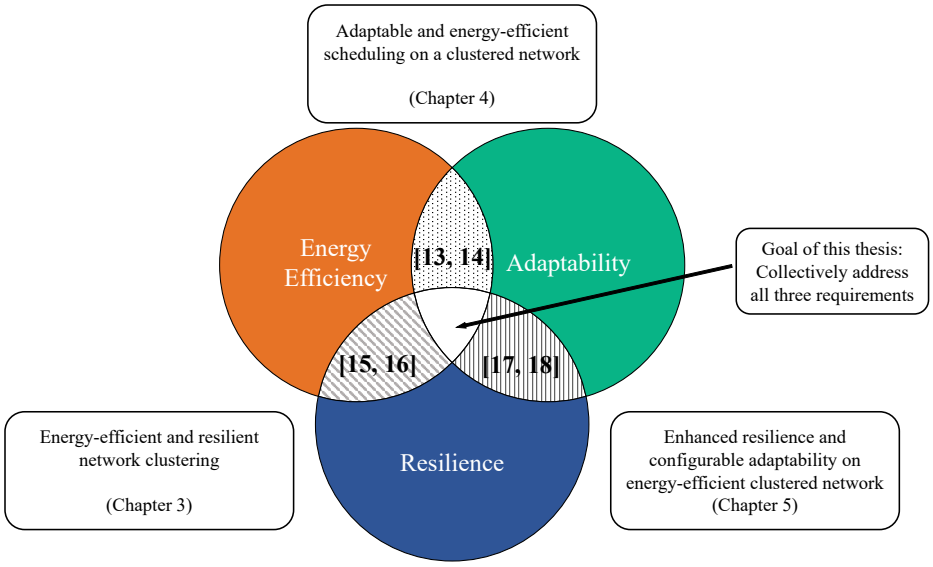


Figure 1.1: Major requirements for an IoT network. Existing solutions meet at most two requirements indicated by shaded area while no solution meets all the requirements.

between the resilience circle with the other two requirements, the available solutions that produce energy-efficient and resilient networks [15, 16] or adaptable and resilient networks [17, 18] reduce dramatically. To the best of our knowledge, there is no existing work to date in the available body of literature that incorporates all the three major requirements as shown by the white area at the center of the figure.

1.2 Challenges to combining the requirements

It remains a significant challenge to collectively address the three requirements of the IoT networks. Here, we discuss the major challenges to overcome in order to meet the requirements.

Orthogonality of the requirements. The requirements of energy efficiency, adaptability and resilience tend to have an orthogonal function with respect to each other. For example, improving the resilience of the network may introduce complex computations arising from additional verification of information which reduces the network's energy efficiency. Similarly, adaptable solutions allow for redundancy and improve the resilience of the network, but require more communication and energy to maintain a common system state across all the end-devices. A fundamental change in the design of IoT networks is necessary to tackle the three requirements collectively.

Resource-constrained devices. Typically, most devices in IoT networks are heavily resource-constrained with processor speeds in the range of a few MegaHertz (MHz),

memory in the range of a few kilobytes (kB) and equipped with a short-range low-power communication standard like Zigbee, Bluetooth Low-Energy (BLE), etc. [4]. In addition, these devices are often powered by batteries with limited capacity in the range of milliwatthours (mWh). Such constraints on resources significantly limit the complexity and the range of communication strategies that can be run on these devices. Hence, it is essential to consider the resource-constrained nature of IoT devices while designing solutions that incorporate all the three requirements.

Changes to network topology. The ad-hoc nature of most IoT networks makes the network topology malleable. Device faults cause holes in the information path - this triggers changes in device roles to restore the information path and hence changes the network topology (*e.g.* failure of the leader (information provider) in the network results in the re-election of a new leader node). Similarly, new devices could increase the number of messages being exchanged to achieve a consistent system state across all devices. The network must adapt to these changes without a significant increase in energy consumption (no. of messages) while maintaining connectivity within the network and the functionality. Hence, a key factor in the design of energy-efficient and resilient communication strategies is to adapt to the dynamic changes in the network.

1.3 Existing solutions address a subset of the major requirements

Existing communication algorithms in IoT applications address one or at most two of the requirements discussed earlier, while none of them simultaneously addresses all three of them. Although adaptability has been addressed independently in various existing solutions [12, 19], we analyze the literature on energy-efficient solutions and resilient solutions that integrate adaptability. This allows for the comparison of existing state-of-the-art that is closest to our proposed solution of collectively addressing all the requirements discussed earlier. This section reviews some of the notable solutions in the literature and their associated shortcomings.

1. **Energy-efficient communication strategies.** There is significant literature on energy optimization in IoT networks. Mathematical optimizers and standard optimization techniques such as linear programming, game theory, etc., are employed offline to adjust the device schedules to minimize cost and energy [20, 7]. These solutions involve complex computations which cannot be applied to resource-constrained devices to handle live messages although they can achieve efficiency and resilience. Similarly, other solutions propose time synchronization protocols to minimize communication and, thereby, improve the energy efficiency of the devices [21, 15, 22]. By having a common notion of time, nodes can turn on their radios only during the communication period and keep them turned off at other times. Most of the existing synchronization solutions use flooding to achieve a common time among the nodes quickly [21, 22]. Flooding involves a central (root) node disseminating the time information that is rapidly spread across the

network, *i.e.,* an incoming message is immediately re-transmitted to all its neighboring nodes. Root nodes in the network also monitor the messages to prevent dissemination of erroneous information, but cannot prevent faults in the root node itself. To prevent a single-point of failure and adaptability issues (multi-hop degradation) in flooding-based solutions, a decentralized synchronization solution also exists [15]. Despite the energy savings from the synchronization, there is a significant communication overhead. The decentralized solutions use redundancy to achieve fault resilience leading to hardware overhead. To further reduce the energy expenditure of the devices, under the assumption of a synchronized network, various *clustering* protocols were designed to minimize the nodes that communicate frequently [23, 24, 25]. Clustering strategies group the nodes into a set of clusters to establish a communication hierarchy; nodes communicate with an elected local node, *i.e.* the cluster head, which in turn communicates with other cluster heads. Clustered network topologies provide a positive trade-off between centralized and decentralized solutions as they alleviate the impact of both, single-point of failure and constant communication. However, none of the existing clustering algorithms focus on maintaining network connectivity in the presence of faulty nodes.

2. **Resilient solutions.** Although decentralized networks provide basic protection against *fail-stop* faults (silent devices) through redundancy, the nodes continue to be susceptible to faults such as spikes, outliers and intermittent transmissions. Fail-stop faults cause a node to be non-responsive permanently, while spikes cause sudden surges in the transmitted data. Outliers are a result of the transmitted information being outside an expected range/threshold. Similarly, transmissions that randomly miss their scheduled intervals cause intermittent transmission faults. Spikes, outliers and intermittent faults can be grouped as a class of byzantine faults that exhibit behavior other than what is expected. A body of work focuses on addressing the aforementioned issues. Synchronization protocols with additional fault handling mechanisms were introduced to minimize the impact of faulty information [26, 27, 16]. Specialized hardware has been used to detect faults in addition to software-based synchronization to improve the resilience of IoT networks [16]. Consensus mechanisms [26] and blockchain [27] are some of the important solutions towards achieving resilience with untrusted nodes in the network. Meanwhile, other solutions use a simple estimation for expected outputs and categorize anything outside the expected result as a fault [28, 26]. Resilience to faulty nodes in the existing solutions comes at the cost of energy efficiency - most solutions require large amounts of processing and communication resources. The fault classification algorithms are memory-heavy while blockchain and encrpytion solutions require significant compute and communication resources making them impractical on resource-constrained devices.

   Addressing the above shortcomings in existing literature, in this thesis, we introduce a novel clustering strategy that incorporates resilience by design in addition to providing adaptable and energy-efficient communication (Chapters 3 and 5). Additionally, we also extract some features of our clustering solution with a use-case from the domain of charge scheduling of electric vehicles (Chapter 4). To the

best of our knowledge, our proposed solutions are the first to collectively address the three major IoT requirements discussed in Section 1.1.

Thesis Statement

This thesis develops communication algorithms to dynamically create a synchronized and clustered network topology in the presence of byzantine device faults, collectively addressing the three major IoT requirements of energy efficiency, adaptability and fault resilience.

1.4 Research Contributions

This thesis proposes various algorithmic solutions to meet the three requirements and demonstrates the gains achieved over existing solutions. In this regard, we make the following contributions:

1. **Dynamic adaptability of network topologies with a resilient yet energy-efficient communication algorithm.** Firstly, a novel communication algorithm called *DeCoRIC* that exploits the ad-hoc nature of IoT networks is established to allow flexible topological changes in a decentralized network of asynchronous nodes. DeCoRIC is the first clustering algorithm to provide resilience against the fail-stop faults, with extensions to other faults in the following contributions. Each node joins the network independently without any prior knowledge of the network parameters (*e.g.* position in the network, neighboring nodes, etc.). The nodes discover themselves and their neighbors through communication. Our proposed solution elects a representative node called *Cluster Head* (CH) in any neighborhood of nodes based on the highest *degree* parameter, *i.e.,* nodes with the highest number of connected neighboring nodes have the highest degree. Other nodes in the neighborhood associate themselves with the nearest CH to form groups called clusters. CH nodes oversee communication within their respective cluster and across clusters when necessary, allowing the remaining cluster nodes to conserve energy. The algorithm also incorporates resilience by design using gossiping [29] and maintains the connectivity among clusters even in the presence of fail-stop faults. DeCoRIC allows for dynamic centralization in the form of clusters based on the instantaneous positions of the nodes while allowing each cluster to operate independently; this method strikes a balance to overcome the shortcomings of existing centralized and decentralized networks.

2. **A real-world use-case demonstrating the applicability of the clustering solution.** Secondly, to demonstrate the applicability of our DeCoRIC with energy-efficient clusters, we examine the problem of load balancing in a smart grid with a network of charging stations and nodes connected to it. We build on a clustered network topology established using certain features (degree of a node) of DeCoRIC where an energy source (aggregator) is a CH and all devices connected to it consume energy within the cluster. To ensure every node gets a chance to consume

energy to meet its energy demand, we develop a low-complexity heuristic solution that provides a feasible charging schedule without the long wait times experienced by optimization solvers. By exploiting the fact that mobile devices can move to different geographic locations (clusters), we show that energy efficiency in the form of utility can be significantly improved. The clustered network and its capability to reconfigure itself amidst mobility of the nodes also ensure that the solution is adaptable.

3. **Accurate network synchronization with network resilience against byzantine faults.** Lastly, we recognize the need for a common notion of time in the network as nodes tend to keep their radios on for a longer duration among asynchronous nodes. To achieve time synchronization, we develop a clustering-based communication algorithm called *C-sync* that synchronizes nodes with a significantly lower energy consumption compared to other state-of-the-art solutions. Every CH node communicates with other CHs to identify their positions in the network. The CH node at the center of a neighborhood called *local center* is selected to disseminate time information to other clusters. The maximum number of hops to the local center is configurable and, hence, ensures that the maximum error in the clock can be bounded. Further, special roles are assigned to the bridge nodes that connect clusters to verify the communicated data and ensure no faulty information is propagated across the network. The fault handling in DeCoRIC is further extended to provide resilience against a class of byzantine faults including spikes, outliers and intermittent communication.

1.5 Organization

The rest of the thesis is organized as follows:

- Chapter 2 provides the necessary background with respect to the communication standards, assumptions and the fault model used for the remainder of the thesis.

- Chapter 3 presents a novel clustering algorithm, *DeCoRIC*, on an ad-hoc decentralized IoT network to achieve connectivity and resilience against faulty nodes. Using the principle of *degree*, Cluster Head (CH) nodes are elected to supervise the communication within a cluster and manage the interaction with neighboring clusters. With the aid of a gossiping mechanism to communicate nodes' states, DeCoRIC provides resilience against failed nodes and recovers network connectivity within a bounded time. We demonstrate that DeCoRIC realizes an improvement of at least 70% in power efficiency and 42% longer lifetimes of the nodes, respectively over the state-of-the-art clustering solutions with random and density-based clustering [23, 24].

- Chapter 4 explores the application of DeCoRIC in a real-world use-case of smart grids. The clustered network adapts well to the distributed energy sources (represented by renewables) to manage the stochastic energy demand of stationary nodes

(*e.g.* household appliances) and mobile nodes (*e.g.* electric vehicles). To maximize the utility (benefit) achieved by all the devices in the network, a distributed online heuristic is proposed to establish a feasible charging schedule. The schedule is created within a practical time (few minutes) while the standard solvers take prohibitive time (few hours) which is impractical for most real-world applications. Experiments on real-world charging data from an electric vehicle testbed [30] show an improvement of over 57% in the utility over standard scheduling solutions such as earliest deadline first and highest demand first.

- Chapter 5 develops *C-sync*, a novel synchronization solution based on DeCoRIC to establish a common notion of time across all the nodes in the network. C-sync is able to achieve μ-second accuracy through energy-efficient communication. Our solution is also adaptable as the number of hops from any node to a time source is bounded. Through atomic broadcasts, a consensus mechanism is introduced to counter faulty nodes. An average reduction of over 56% in the power consumption of nodes is achieved compared to the existing state-of-the-art synchronization solution based on neighbor-clock averaging [15]. C-sync is tested on a real-world testbed and shown to be resilient against a subset of byzantine faults (spikes, outliers, stuck-at and intermittent communication faults) as byzantine faults encompasses a wide range of faults [31].

- Chapter 6 concludes the thesis by summarizing the contributions and provides future opportunities on the presented research.

2 Background

In this chapter, we introduce the preliminaries used for communication, assumptions, network model and simulation platforms used throughout this thesis. We also provide the details of all the fault models that are addressed by our fault-resilient solutions. These details provide the foundation for the proposed solutions discussed in Chapters 3, 4 and 5.

2.1 Communication topologies

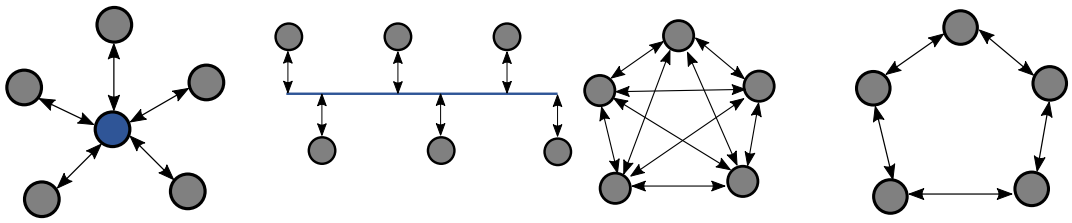
Communication topologies (architecture) can be realized in different forms based on the design of the communication algorithm used, *i.e.,* the rules of communication and their available neighborhood information for each device. The arrangement of nodes (topology) and the processing entity also influence the communication architecture. Traditionally, the communication topologies in the literature are categorized into centralized and decentralized networks [32] and are shown in Figure 2.1.

2.1.1 Centralized network

In a centralized network, multiple devices/nodes collect data and send it to a central node (root node) for processing. All nodes in the network communicate with each other through the root node and take actions based on data/commands received by the root node. During initialization, every node needs to be configured with the root node information such that the topology is maintained. The central node in the network handles the bulk of all data processing and communication in the network. Common topologies used in centralized networks include star and bus topologies shown in Figures 2.1a and 2.1b, respectively. Furthermore, there is an additional overhead of a complete network reconfiguration in the event of a network change (due to node failure, node mobility, etc.). A failure of the central node in a star topology or a failure of the main communication link in the bus topology necessitates a re-setup of all the devices in the network. Network changes are common in ad-hoc IoT networks due to their plug-and-play property [33].

2.1.2 Decentralized network

Each node operates independently in decentralized networks with data exchange among neighboring nodes. Network decisions are made through the collective agreement of all nodes in the network. Nodes in a decentralized network require no prior knowledge of other nodes in the network since they function independently and communicate with



(a) Star Topology **(b)** Bus Topology **(c)** Mesh Topology **(d)** Ring Topology

Figure 2.1: Common topologies used in IoT networks.

the neighboring nodes to gather information about the network. Mesh and ring topologies are commonly used in decentralized networks as shown in Figures 2.1c and 2.1d, respectively. Since each node operates independently, there is no necessity for network re-configurations as seen in centralized networks. Hence, a failure results in configuration changes only among the neighbors of the failed node, without impacting the other parts of the network. However, since the collective decision of the nodes is made by exchanging messages among each other, multiple nodes communicate in parallel. This results in significant interference on the communication medium/channel among the nodes leading to loss of messages [15].

2.1.3 Clustered network

Both the centralized and decentralized networks are inadequate to provide the requirements of energy efficiency and adaptability discusssed in Section 1.1. On the one hand, centralized architectures offer an efficient mechanism of communication between the nodes, but suffer from a single-point of failure and reconfiguration overheads. On the other hand, decentralized architectures allow ad-hoc changes to the nodes and support network adaptability, but experience a high load of communication and channel interference. To achieve an efficient operation and communication among nodes (offered by centralized architectures), while retaining the adaptable ad-hoc capabilities of a decentralized network, a clustered communication network where nodes form pockets of centralized networks within an overall decentralized network is used [23, 34, 24, 25].

To begin with, every node is independent, similar to a decentralized network. However, a key difference is that these individual nodes communicate with their neighbors to group themselves into clusters. Nodes across different parts of the network form multiple clusters in their neighborhood. The nodes in each of the clusters elect a representative node among them to oversee the communication within the cluster as well as handle information exchange across different clusters. An example of a clustered architecture is shown in Figure 2.2 where the nodes in orange represent the leader nodes called Cluster Heads (CH), while the nodes in grey are Common Nodes (CM) within the cluster.

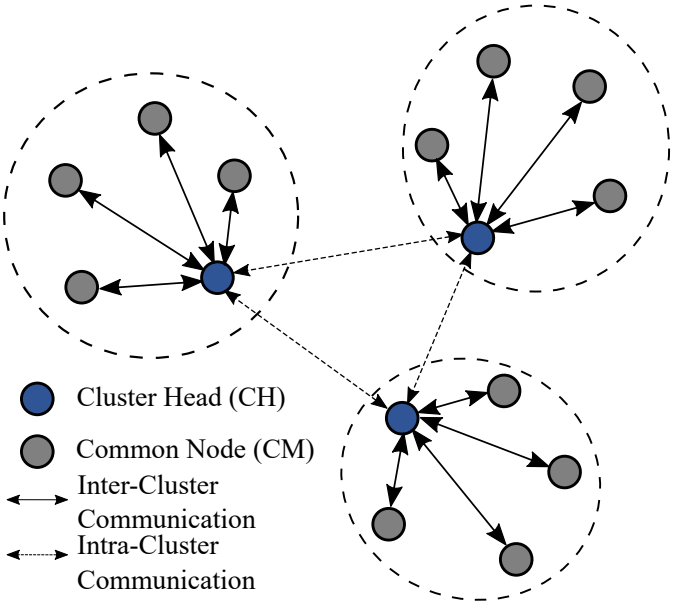


Figure 2.2: A clustered network combining the properties of centralized and decentralized networks.

2.2 Communication primitives

We assume that communication across all nodes of the network is on a wireless medium due to their low costs, high availability in the market and fast-evolving speeds that match or exceed the wired counterparts. Unless otherwise stated, the IEEE 802.15.4 communication primitive is used as a standard for the interaction among the nodes [3]. The standard was introduced in 2003 for low data rate wireless networks which gave rise to various protocols such as ZigBee, WirelessHART (Highway Addressable Remote Transducer Protocol), 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks), DSSS (Direct Sequence Spread Spectrum), etc., that differ in their implementations from layer 3 (Network layer) and above on the OSI stack. In this thesis, we adopt the 802.15.4 standard with a focus on the common lower layers, operating in the frequency range of 2.4GHz with offset-quadrature phase-shift keying (O-QPSK) modulation scheme. Europe and North American regions have different frequency ranges with the same spectrum depending on the release/version of the protocol being used.

2.2.1 Frame Structure

The frame structure with Physical and MAC (Medium access control) layers are shown in Figure 2.3. At the physical layer, the preamble sequence identifies the IEEE 802.15.4 protocol. The start of frame delimiter is used to recognize the start of frame transmission/reception to/from the channel at the Physical layer. We use this in Chapter 5 to find the precise time of data transmission at the physical layer. Frame length indicates

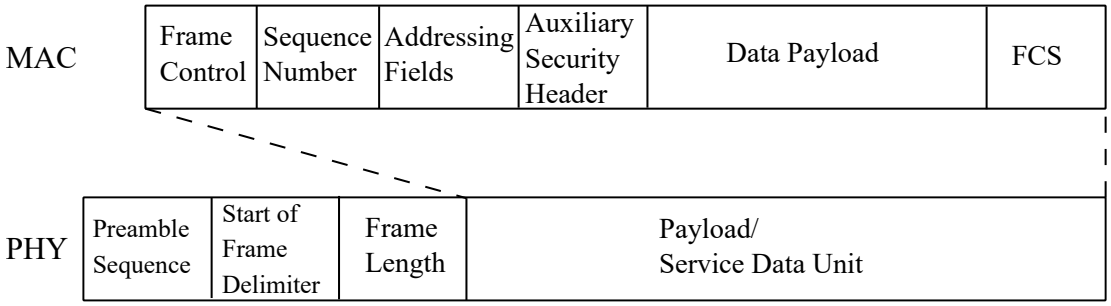


Figure 2.3: The frame structure of the IEEE 802.15.4 protocol at the Physical and medium access control layers.

the number of bytes present in the payload/service data unit where the payload stores the frame from the MAC layer.

The MAC layer controls the channel access mechanisms for the physical layer. The frame control field separates different types of frames such as a data frame, an acknowledgment frame, a beacon frame or a command frame. The sequence number ensures acknowledgment frames are associated with the correct data frames and that beacon/-data frames are sent in a correct continuous sequence. The source and destination addresses are specified in the addressing fields. Broadcast messages do not have any destination address. If the frame is configured to be secure, then the associated security data such as keying mechanism details are stored in the auxiliary security header field. The data payload stores all the data to be sent/received to/from the destination device. Contents stored within the payload can be configured to suit the application. Frame Check Sequence (FCS) ensures the integrity of the frame by storing the cyclic redundancy check (CRC) to detect any errors.

2.3 Network Model

The IoT network can be treated similar to a graph. A Graph G can be defined as a pair given by $G = \langle V, E \rangle$ where V is the set of vertices in the graph and $E \subseteq V$ x V is the set of edges between the vertices. For any two nodes $u, v \in V$ if the edge $e_{uv} \in E$ and $e_{vu} \notin E$, indicating a unidirectional edge from u to v, such a graph is called directed graph or digraph. If nodes u and v have an edge e_{uv} such that $e_{uv} = e_{vu} \forall e_{uv} \in E$, the edge is bi-directional; a graph all edges exhibiting this property is called an undirected graph. If there is a path/cycle from a vertex v traversing back to itself, the resulting graph is cyclic. Similarly, acyclic graphs do not have any cyclic paths in the graph. The different graph types are shown in Figure 2.4.

Based on the assumptions listed in Section 2.3.1, the nodes are connected to each other in a wireless network with duplex communication with bidirectional links. Additionally, the nodes at the edge of the network do not have any cyclic communication links. Hence, we consider our network to be an undirected acyclic graph. The nodes of the network

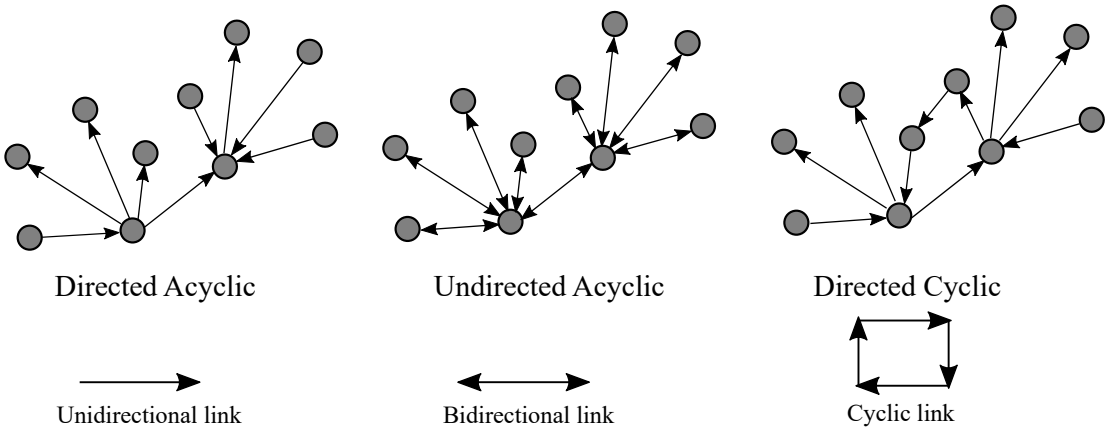


Figure 2.4: Different types of Graphs based on the direction of communication links among the nodes.

operate independently while taking collective decisions to manage the network structure (*e.g.* clustered network).

2.3.1 Model assumptions

A set of general assumptions are made that are applicable throughout this thesis. The assumptions not only provide details on the overall network setup but also provide details on the behavioral traits of the nodes. Any changes or additional assumptions specific to a solution will be listed in the specific chapter (in Chapters 3, 4 or 5). The list of assumptions are as follows:

1. An ad-hoc network of devices is assumed where each device takes independent decisions by communicating with neighboring devices without apriori knowledge on the network topology (*i.e.,* fully decentralized network).

2. Nodes can join or leave the network at runtime (e.g. node failures or new node additions)

3. Devices are heterogeneous, *i.e.,* each device can be functionally different without a common manufacturer, sensors, etc.

4. All devices are equipped with limited resources including a processor, clock, memory and a unique identifier (ID) for recognition and tracking.

5. All nodes transmit and receive on the same channel.

6. The communication protocol among the devices is based on IEEE 802.15.4 standard [3] unless otherwise specified.

7. Carrier-sense multiple access with collision avoidance (CSMA/CA) is used at the MAC layer of the protocol stack to prevent interferences during communication.

8. There are no fixed topologies or arrival/departure times of devices unless specified with nodes moving in and out of the network at any position/time.

2.3.2 Fault model

In this section, we define the fault model adopted from the literature in this thesis [35, 36, 37]. One or more of the faults described below are addressed in the solutions described in Chapters 3, 4 and 5. The type of faults considered in this chapter are as follows:

Fail-stop faults. Fail-Stop faults in a device render it completely non-functional, *i.e.,* the device becomes *silent* and stops communicating. This scenario is equivalent to the absence of the device and could create network partitions if the device has no backup in its information path. Such faults are caused by hardware damage, environmental factors such as earthquakes, avalanches, battery exhaustion, tampering, etc. Fail-stop faults can be critical especially if the fault occurs on the central entity of a system as it mandates a network-wide re-configuration of all devices to choose the new central entity.

Stuck-at faults. Stuck-at faults make the devices partially functional, where the device transmits a constant value independent of any changes in the environment/input. The fault could be caused due to physical damage or a software bug. Although the devices can receive data, they are able to output only a single value. Stuck-at faults can be very hard to detect as the constant value may match the ground truth a few times. This leads to false data reporting and damage to the system.

Spike faults. Spike faults are sudden increases or decreases in the communicated values and subsequent reports of normal values. Other neighboring devices could react to the spike fault and make the system overloaded. Although spike values are important in applications such as environment monitoring (e.g. volcanoes and earthquakes), most applications do not expect a sudden increase in value at all times. Spike faults can be detected by setting a threshold on expected values and comparing them against other neighboring devices.

Outlier faults. Outliers are similar to spike faults with a distinction of less extreme variations in its output. Outliers can be particularly hard to detect as their communicated values are typically very close to the boundary of expected values. Similar to spikes, outliers are detected using a threshold on the expected values. However, outlier faults consistently report values close to the threshold that can be used for detection.

Intermittent communication faults. A device with an intermittent communication fault is unable to establish a communication channel with its neighboring devices consistently. There are irregular periods when the device does not report any value while managing to report sometimes. With synchronization in the network, the faulty device

randomly misses its schedule to transmit. Any device which does not report values consistently for a configured number of times can be classified to be faulty with intermittent communication.

Byzantine faults. Byzantine faults are a broad category of faults that includes all of the above faults and more. A device is considered as byzantine faulty if its behavior does not follow the expected behavior [38]. This category of faults includes a faulty device which may or may not coordinate with other faulty devices in the system. The devices are partially functional and present different behavior to different devices and at different times. It is very difficult to identify such faults as the coordination among the neighbors could fail due to different observations by each device.

2.4 Simulation Platforms

2.4.1 Contiki Operating System

Contiki is an operating system designed specifically for inter-connected resource-constrained wireless IoT devices of a network [39]. The OS has a modular design with a TCP/IP stack as well as a lightweight proprietary Rime stack designed specifically for resource-constrained wireless IoT devices.

Contiki MAC and X-MAC

Contiki MAC. Contiki MAC is the default MAC layer used in Contiki for low-power operation with very low-powered listening [39]. The duty cycle of the radio can be as low as 1% and has the highest power efficiency among all the other available MAC protocols in Contiki. To achieve the high power efficiency, the timing for the packet transmission must satisfy certain constraints listed as follows:

$$t_i < t_c, \text{and}$$
$$t_r + t_c + t_r < t_s$$

where t_i is the packet transmission interval (time between two packet transmissions) , t_c is the interval between two channel checks (channel check for availability to transmit), t_r is the time required to obtain a stable Received Signal Strength Indicator (RSSI) value (time to get a good quality signal with low noise) and t_s is the minimum transmission time of the smallest packet (time between sender sending the last bit of packet to receiver receiving the same).

The first inequality of the constraint ensures that the CCA can recognize at least one packet transmission. A minimum packet size is ensured by the second inequality of the above constraint to prevent packets from being missed in the interval between two consecutive CCAs.

Incorporating the constant values from the IEEE 802.15.4 standard [3], we obtain,

$$0.352 < t_i < t_c < t_c + 0.384 < t_s \tag{2.1}$$

Contiki X-MAC. The Contiki X-MAC layer is based on the original X-MAC protocol with improvements on phase awareness (sleep-awake cycles) of the nodes and adaptation to the IEEE 802.15.4 protocol [40]. As there is a periodic wakeup among unsynchronized nodes to check for incoming messages in the X-MAC protocol, the change in phase due to lack of synchronization is used by the Contiki X-MAC to minimize power consumption when the phases are aligned. Contiki X-MAC also allows streaming for packets to be sent in quick succession and enables the receiver to keep the radio on for incoming packets.

In order to achieve phase awareness, Contiki X-MAC uses the *rtimer*, contiki's real-time timer module, to get the exact time stamps *i.e.,* when a packet is received or sent, the exact time is recorded when the counter node had its radio on. Using the time information, the phase can be estimated to achieve a loose synchronization between the sender and receiver. Furthermore, a function named *powercycle()* is used to switch the radio on and off based on the fixed schedule created using the rtimer.

Cooja Simulator. Contiki's Cooja Simulator (Java-based) allows development in native C language, which can then be directly deployed on a compatible hardware platform [39]. The software elements are cross-compiled to target hardware, similar to an emulation flow. This enables the evaluation to consider actual hardware constraints such as memory limitations (to fit the algorithm), network errors such as packet loss and interference, and actual bit-level transmission at the cost of slower execution time. If the memory footprint of the algorithm is large, it will not fit within the node's limited resources. There is a real-world network traffic simulation with packet loss and interference, as nodes are emulated at a hardware-level. Although the simulation might slow down due to the hardware mapping, it gives a more precise estimation of parameters. The same firmware can directly be loaded into physical devices.

Hardware Platform. As we assume Skymote [4] for most of our experiments, we discuss the available timers in the hardware. The underlying MSP430 microcontroller comprises two clock sources: a Digitally Controlled Oscillator (DCO) and a crystal oscillator. We employ the *powertrace* tool in Contiki to measure the power consumption of the devices for all our experiments [41]. Skymote uses a Texas Instruments CC2420 transceiver that complies with the 2.4 GHz IEEE 802.15.4 6LoWPAN standard with a bit rate of 250 kbps and a processor platform that supports sustained low-power mode. The hardware also supports a sustained low-power operating mode for the CPU. We also tested the Z1 mote [42] as target hardware to verify the capability of the algorithm to work in a heterogeneous environment. The crystal produces a stable output at 32,768 Hz while the DCO produces a 524,288 Hz output that is highly susceptible to drifts from voltage (0.38%) or temperature variations (20%). Two timers A and B are driven by one of the two clock sources independently, with a 16-bit output register.

Timers. The crystal oscillator with a resolution of 30μs is unsuitable for WSN applications as they require accuracy in the order of microseconds or higher. DCO at a resolution of 2μs is more suitable to achieve high accuracy applications such as clock synchronization, etc. In an existing time synchronization protocol [22], the authors

minimized software instructions to reduce the impact of DCO instability on the timer output. Based on this literature, we can make use of a crystal oscillator to stable the DCO output by mapping the ratio of frequencies between the two sources. By measuring the difference in DCO ticks at every crystal oscillator tick, we can measure the deviation in the clock and correct the drift.

3 DeCoRIC: Energy-efficient, connected and resilient clustering

As discussed in Chapter 1, information exchange between IoT nodes over a wireless medium requires the design of energy-efficient communication strategies to ensure prolonged and sustainable operation of the devices under different and often challenging conditions. Additionally, in critical systems such as industrial manufacturing plants, it is vital to have active maintenance of the communication links so that the data can be relayed from one part of the network to other parts reliably. The presence of safety issues, network volatility, security flaws or energy exhaustion pose a challenge in maintaining this end-to-end connectivity among all nodes of an ad-hoc network. IoT networks must re-adapt communication links for any changes in the node configurations to ensure connectivity. Hence, IoT networks not only need to be energy-efficient but be resilient to the failure of nodes or changes in topology.

Clustering has been shown as the most effective technique to improve energy efficiency and effectively handle device dynamism in networked systems [23]. Clustered networks provide the necessary trade-off between centralized and decentralized networks in terms of energy consumption and adaptability. They incorporate the energy-efficient operation of centralized networks while avoiding the single-point of failure. Similar to decentralized networks, clustered networks allow devices to join or leave the network in an ad-hoc manner without impacting the network performance.

Nodes, as illustrated in Figure 3.1 (a), are grouped into *clusters* based on common node properties such as residual energy, location or degree (number of communication links of the node). Cluster sizes can be equal or unequal depending on the chosen property. Nodes marked in blue are elected representative nodes called cluster heads (CHs), each of which acts as the data aggregator and nodal point for multi-hop communication, as shown in Figure 3.1 (b), allowing regular (non-CH) nodes to operate in low-power mode more often to conserve energy.

Existing solutions in the literature have addressed the problem of clustering by utilizing various node properties such as degree, remaining energy of the node, etc. However, the problems of overcoming network partitions due to faults and maintaining end-to-end connectivity remain a problem to be solved. Also, static topologies (fixed roles for nodes) lack the flexibility to deal with the ad-hoc nature of an IoT network as well as with node failures during operation. Such networks are unable to accommodate new nodes or deal with the failure of nodes at run-time. Hence, in addition to the capabilities of the existing clustering techniques, we believe that the following properties are essential in any IoT clustering technique:

1. **Connectivity** – the clustering technique must ensure that the nodes are clustered such that there is a path between any two nodes in the network whenever possible;

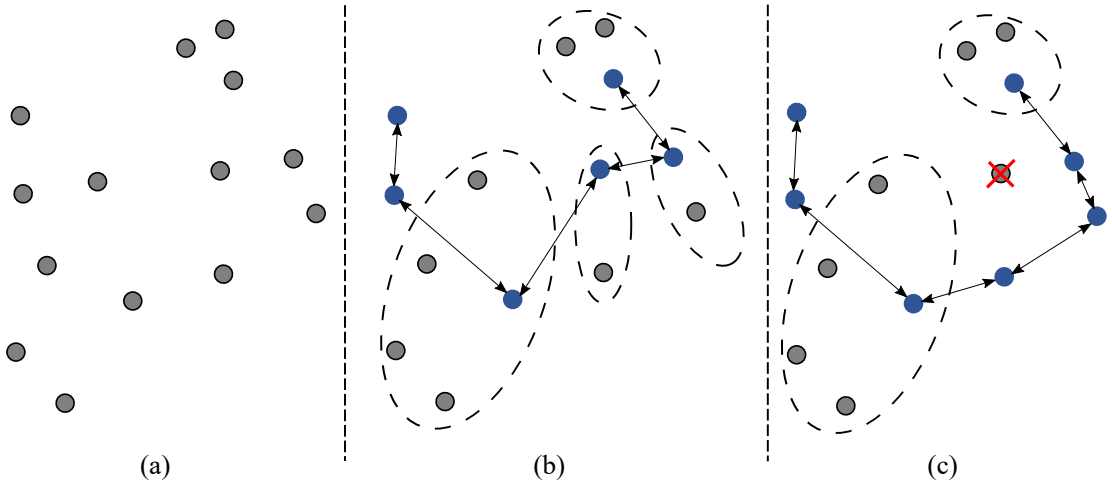


Figure 3.1: Clustering operation: starting with any topology (a), the nodes align themselves into clusters with an elected CH (b). With DeCoRIC, the clustering dynamically adapts to changes in topology to ensure connectivity among all nodes (c).

this property ensures reliable routing of information between any two nodes in the network.

2. **Adaptability** – there is no central entity managing connections across the network; devices make independent decisions restore connections without impacting the network functionality.

3. **Resilience** – the network must adapt to node faults or network changes at runtime by detecting and reorganizing in a time-bound manner to ensure *connectivity*.

In this chapter, we present Decentralized Connected Resilient IoT Clustering (DeCoRIC), a clustering scheme that can group nodes into connected clusters and adapt to network changes at runtime without relying on a central node or prior information (topology, position, etc.). Through DeCoRIC, we aim to collectively address all of the above three properties of connectivity, adaptability and resilience in an energy-efficient way for any IoT network. DeCoRIC starts with an ad-hoc asynchronous and decentralized network where nodes do not have any prior information about their position or the network topology. Nodes communicate with the neighboring nodes to familiarize themselves with their position in the network. Based on the information gathered from their neighbors, nodes make decisions and react to dynamic topology changes by altering their state, as seen in Figure 3.1 (c), to ensure connectivity, while minimizing energy overheads. While existing schemes handle failures implicitly through regular re-clustering, DeCoRIC has an explicit failure handling mechanism and strives to achieve connectivity in a power-efficient way. DeCoRIC can react to the failure of critical nodes like CH's, non-CH low-power leaf nodes or other network changes without requiring centralized management. Also, it ensures that each cluster is connected to at least one other neighboring cluster as long as the nodes are within radio range, assuring that (critical)

information from any node can be accessed by any other node in the system. The connectivity allows the deployment of routing protocols for various applications. DeCoRIC achieves these goals while offering improved energy efficiency to existing clustering techniques.

LEACH [23] and BEEM [24] are chosen as the representative schemes for comparison. LEACH is the de-facto benchmark of clustering algorithms, while BEEM is a recent extension of another benchmark, Hybrid Energy-Efficient Distributed clustering (HEED) [25], aimed at higher connectivity. We evaluate DeCoRIC using multiple random network topologies to show that the above properties are achieved.

This chapter presents the following contributions:

(i) We discuss the related solutions in the literature and provide a comparison between DeCoRIC and other prominent solutions in Section 3.1

(ii) We propose DeCoRIC, our scheme for Decentralized Connected Resilient IoT Clustering and its implementation details in Section 3.2.

(iii) In Section 3.3, we have implemented the state-of-the-art techniques LEACH and BEEM into the Contiki simulator to emulate a realistic communication environment for comparison with DeCoRIC and made the implementations open-source for the community.

Further, we show through results that DeCoRIC converges to a resilient fully connected network with bounded latency and achieves up to 110% and 70% higher power efficiency and 109% and 42% longer lifetime compared to LEACH and BEEM, respectively.

(iv) We summarize the contributions of DeCoRIC in Section 3.4.

3.1 Related work

Radio communication is a key component that largely influences the energy consumption in IoT nodes. Clustering techniques aim at reducing this energy consumption by grouping neighboring nodes into clusters. Each cluster has an active cluster head (CH) as a representative node elected by either a central entity or by all the nodes in the cluster. Clustering enables the regular (non-CH) nodes to reduce the periodicity of transmission and operate in low-power mode, reducing the overall power consumed by the system. The cluster head oversees the multi-hop communication and performs data fusion resulting in minimal communication for the non-CH nodes. Clustering techniques can be classified into centralized and decentralized methodologies, based on whether the clustering decision and CH election are performed by a central entity or independently by nodes of the network.

Centralized clustering. Centralized techniques rely on a central entity that has global knowledge of the network and manages the CH election and clustering process. The clustering operation can be based on the degree of a node in the network [43, 44], residual

energy of nodes [45, 46] or other parameters. The clustering problem was formulated as a linear programming problem in [47], representing a trade-off between energy consumption and the quality of communication. A centralized version of a popular decentralized algorithm, LEACH (described below), was developed in [34], where control decisions are managed by a central entity, making more efficient CH selection than LEACH. Further improvement was made in [48], where nodes are organized into a chain based on proximity to evenly distribute the transmission energy. While there is no deterministic polynomial algorithm that can partition a network topology into clusters [49], metaheuristic algorithms like particle swarm optimization and artificial bee colony have been successfully applied in the clustering of wireless networks [50, 51]. Bee-Sensor-C [52] implement a localized event-based clustering by grouping nodes around an event (*e.g.* change in sensor value); sensors that are not in proximity to the event do not form clusters resulting in low energy efficiency. The requirement of a central entity (often the base station) in centralized systems results in higher clustering latency and adaptability issues since every decision has to be relayed from the central entity. The centralized approach also results in a single-point of failure at the central node, inhibiting effective ways to enable resilience and connectivity. Although centralized techniques generally offer superior energy efficiency, distributing the clustering operation among nodes aims to mitigate some of these issues.

Decentralized clustering. HEED was one of the earliest decentralized techniques and uses a combination of node degree and residual energy as the metric for clustering [25]. In the HEED clustering algorithm, the residual energy parameter creates a high density of Tentative CHs which eventually become CH or non-CH nodes. BEEM [24] follows similar steps as HEED with an exception of CH Election process that includes the degree of a node in addition to HEED's residual energy and probabilistic election. This improves connectivity by letting nodes in denser areas expend higher energy. PASCAL improved the power efficiency of HEED through multi-level sectoring [55].

Low-Energy Adaptive Clustering Hierarchy (LEACH) [23] is the most popular decentralized clustering technique, that uses a probabilistic election for the CH nodes. The role of CH is rotated among the other nodes of the cluster to ensure uniform energy distribution across all cluster nodes. Enhancements to the LEACH protocol that enable power optimization through two-level adaptive clustering [56], and multi-level hierarchical clustering [13] have also been proposed. More recent enhancements to the protocol added residual energy [46] and multi-hop communication [57] in the CH election process to achieve minor improvements in energy and throughput, respectively. [46] creates a 2-stage process with the first stage reusing LEACH and the second stage adding residual energy information to update the CH.

Other notable works include overlapping clusters [58, 59] where nodes belong to multiple clusters simultaneously to ensure connectivity among the clusters. Research has shown that unequal clusters tend to have better energy efficiency and can handle a large volume of data transmissions [60]. Hence, unequal clusters [54] were used to reduce the impact of high activity for nodes close to the base station. The work in [53] form multi-level clustering using overhearing characteristics of the wireless medium to form clusters

Table 3.1: Comparison of notable works in Literature.

Property/ protocol	LEACH [23]	PEGASIS [48]	HEED [25]	PEACH [53]	EEUC [54]	BEEM [24]	DeCoRIC
Location/ topology data	No	Yes	No	Yes	Yes	Yes	No
Centralized/ Decentralized	Decentralized	Centralized	Decentralized	Decentralized	Decentralized	Decentralized	Decentralized
Complexity	Low- $\mathcal{O}(N)$	High- $\mathcal{O}(N^2)$	Low- $\mathcal{O}(N)$	High- $\mathcal{O}(N^2)$	Low- $\mathcal{O}(N)$	Low- $\mathcal{O}(N)$	Low- $\mathcal{O}(Degree)$
Clustering mechanism	Residual energy	Location	Residual energy	Proximity (overhearing)	Residual energy and Base station proximity	Residual energy and Degree	Degree
Communication channel	TDMA	CDMA	TDMA	TDMA	TDMA	TDMA	CSMA
Connected clusters	No	No	Yes	No	Yes	Yes	Yes
Resilience to failures	No	No	No	No	No	Yes	Yes

adaptively. Algorithm for Cluster Establishment (ACE) [59] also employs an overlapping strategy for connectivity with CH election based on the degree of node and supports for ad-hoc network architecture creating high overlap among the clusters. Ring-Structured Energy-Efficient Cluster Architecture (RECA) relies on pre-elected CHs to achieve energy efficiency by adding a dynamic re-clustering scheme that alters the clusters if the energy falls below a preset threshold [61]. Few methods use multi-hop from CH to the leaf nodes causing higher node activity. Methods such as [13] use multi-hop within the cluster, causing high node activity and energy consumption, while also presenting challenges in reliable message delivery and clustering convergence when the network scales. Techniques that employ overlapping for connectivity [58, 59] are susceptible to hidden node collision faults, affecting reliable message exchange. However, most of the existing decentralized schemes use a fixed network topology and cannot cater to dynamic ad-hoc networks of the IoT. Further, most techniques do not consider connectivity across all nodes and often result in isolated clusters, albeit the nodes are within the radio range of each other. A summary of some works and their properties is shown in Table 3.1.

Additionally, there is a body of literature that looks into clustering from a graph theoretic perspective [62, 63, 64]. The network is mapped as a unit disk graph to find the minimum connected dominating set (MCDS) for various network topologies. However,

the literature in this direction is unrelated to the presented algorithm in this chapter as they do not consider any radio model in the network, leading to a theoretical solution that may not be practically viable due to different assumptions of the model.

To the best of our knowledge and as observed in the literature [32], there is no existing clustering method that combines the three properties of decentralization, connectivity and resilience. DeCoRIC addressed these challenges while offering comparable energy performance to the centralized clustering schemes.

3.2 DeCoRIC strategy

In this section, we will introduce the detailed clustering states of DeCoRIC. It is important to note that the network uses a fail-stop fault failure model, i.e., a faulty node ceases to transmit information on the network.

3.2.1 Radio primitives

3.2.1.1 CSMA-CA

The MAC layer of the communication stack ensures that the channel is sensed before starting a transmission. Two methods used in this layer to prevent collision are Carrier Sense Multiple Access (CSMA) in the form of Collision Detection (CSMA-CD) and Collision Avoidance (CSMA-CA).

CSMA-CD uses a simpler implementation by detecting a collision in the channel and achieving faster recovery time and preventing further collisions. A re-transmission is triggered to ensure every node detected the collision and starts a back-off before sending the next data packet. It is used primarily in wired networks like Ethernet and is a costly operation for wireless networks in terms of power and transmission time.

CSMA-CA is used in wireless networks and uses the listening mode of a device to ensure the channel is available before transmission. Additionally, the handshake messages-Request to Send (RTS) and Clear to Send (CTS) assist in collision avoidance by confirming a channel clearance before starting the transmission. The total time taken to access the channel is called the clear channel assessment time (τ_{cca}).

In our case, due to the use of a wireless channel, the MAC layer uses CSMA-CA to prevent interference during channel access. The device introduces a random delay based on a configurable parameter called the *back-off* period if the channel is busy or occupied. Before any transmission is made, a Clear Channel Assessment (CCA) is carried out with a time τ_{cca} to ensure that the channel is available for use. The transmission is made at the boundary of a transmission slot after a random number (capped by the configuration) of back-off intervals. The back-off interval is configured by changing the value of the back-off exponent (BE) parameter. Thus, the total delay for back-off ($\tau_{\text{bo,i}}$) is in the range $[0, 2^{BE(i)} - 1]$ where i indicates the count of number of retries up to a maximum of $maxR$. This process prevents interference when multiple devices try to access the channel at the same time.

3.2.1.2 Radio duty cycling

Radio communication consumes the highest power in wireless sensor devices [23]. Due to high energy requirements, keeping a radio on all the time for communication could deplete the battery completely within a short period of time. To achieve a longer lifetime for devices, the radio resource has to be used sparingly *i.e.,* the radio must be switched off when the device is not sending or expecting to receive a message. In this regard, the turning the radio on/off has to be made a periodic affair since the device may not know the exact time of reception without synchronization. The period, RDC_{rate}, is a configurable parameter that is set based on the activity of the radio in the network. Synchronization among devices allows the establishment of a schedule in the duration when the radio is turned on. Conversely, without synchronization, a periodic switch to the radio is employed based on the neighboring devices such that the transmission and reception of the messages are maximized.

3.2.2 Clustering state machine

DeCoRIC operates independently at each node in four states defined by a Finite State Machine (FSM) as shown in Figure 3.2. Each state represents the operating condition of a node in relation to its network and lasts for a pre-configured period called *round* (discussed in Section 3.2.2). All nodes power up in the *Discovery* state and run the same algorithm, where each node waits to receive messages from its neighboring nodes and evaluate its environment. The transition occurs at the end of the period to the *Election* state, where the node with the highest degree declares itself as a CH followed by neighboring nodes associating themselves to nearby CHs to form clusters. The degree of a node is defined as the number of neighbors within its radio range capable of establishing communication. Progression to the *Correction* state after the Election state initiates evaluation of the connectivity property to identify isolated clusters/nodes. Non-CH nodes within a cluster, which can enable connectivity between two CH nodes that are out of range, break out to form *Bridge-CH* nodes to prevent network partitions. The system transitions into the *Stable* state at the end of the Correction state, where the nodes periodically check the status of their neighbors by exchanging *health* information. In the Stable state, a Bridge-CH could upgrade itself to a CH, if newly joining nodes have better proximity to the Bridge-CH and affiliate themselves with the Bridge-CH, forming new clusters. If changes are detected (i.e., failures or new nodes in the system), the nodes go back to the Election state and follow the path to re-establish a stable operation.

To enable this operation, DeCoRIC uses broadcast messages as payload, shown in Figure 3.3, encapsulated in the IEEE 802.15.4 frame. The *Node ID* field marks the ID of the transmitter and is always present in all messages. Only the Node ID field is valid in the Discovery state as there is no information about the neighboring nodes. In the subsequent states, the *CH ID* and degree become known to each node, which feeds into the *neighbor list* (i.e., a list of all neighbors a node has received direct messages from). In different states, parts of the frame get filled based on the information each node has about its neighbors While the neighbor list contains all neighbors that are in the range of

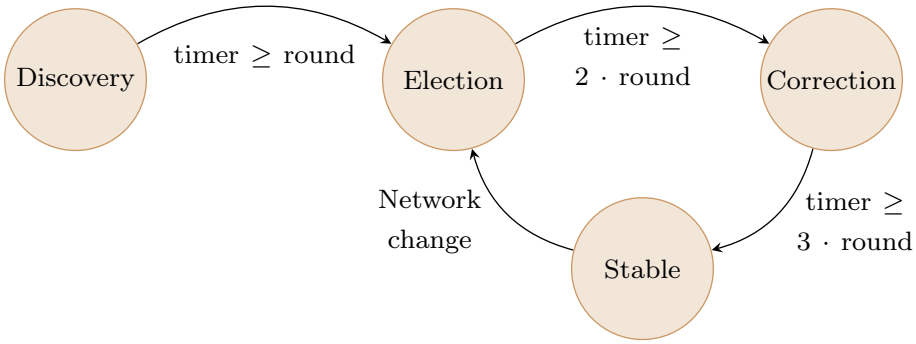


Figure 3.2: DeCoRIC states and transition conditions.

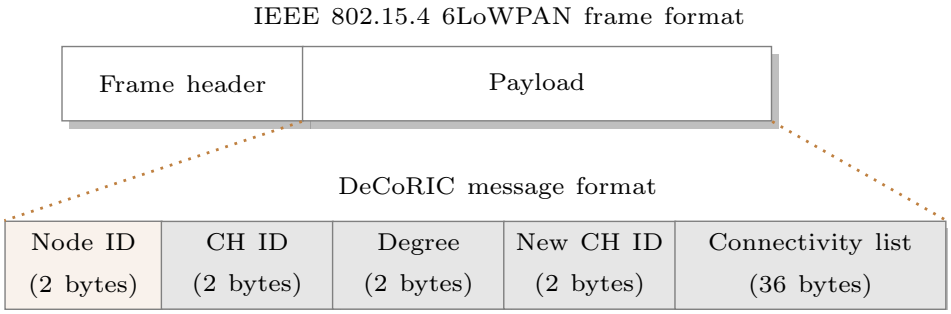


Figure 3.3: DeCoRIC message format.

the node, the connected nodes are maintained using a second list called the *connectivity* list. The connectivity list gets updated periodically with every new message received, reflecting the activity of connected nodes. This two-level list structure allows DeCoRIC to eliminate false positives on the propagation of the activity of the nodes during the Stable state (discussed in Section 3.2.2). The *New CH ID* field is used only when a new node determines that it has to be the CH as it attained a higher degree than its current CH. The operation details of each node as it transitions through the DeCoRIC states are described below.

Discovery state

The neighbor discovery state enables each node to discover the neighboring nodes that it can communicate with and, hence, its own degree. The steps involved in the Discovery state are listed in Algorithm 1. In this state, each node sends a DeCoRIC *ping* message with only the node ID and CH-ID fields filled with its own identifier (others left as zeros). All nodes keep their radio active during this state to receive messages from their neighbors.

The RSSI threshold is a configurable parameter to ensure that communication links among the nodes can offer sufficient signal-to-noise ratio (SNR) [65] for reliable commu-

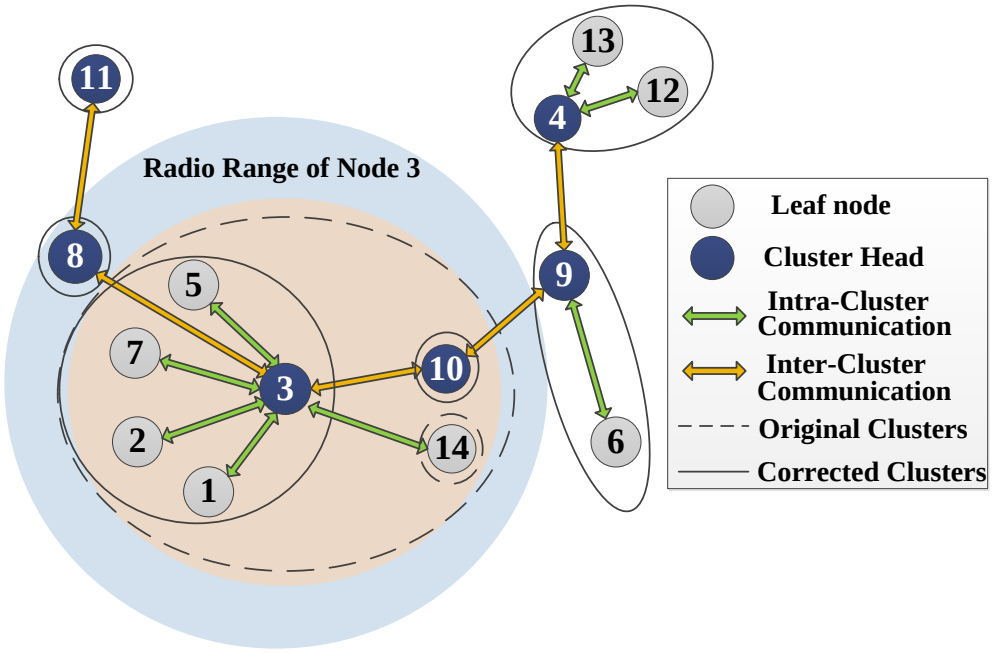


Figure 3.4: DeCoRIC on an example network (not drawn to scale).

nication. A receiving node updates its degree and the neighbor list with each received message. Nodes that meet a received signal strength indicator (RSSI) threshold are marked as *potential* neighbors that could belong to the same cluster. Nodes that fail to meet the threshold do not belong to the same cluster and are marked as external neighbors. In both cases, each receiving node saves the ID of the transmitter to the neighbor list.

For the example network shown in Figure 3.4, node 3 has a communication range of the blue shaded area while the RSSI threshold limits the cluster range to the orange shaded area. Node 3 receives messages from nodes 1, 2, 5, 7, 8, 10 and 14 leading to a degree of 7. Node 8 is marked as an external neighbor with its received RSSI outside the configured threshold while nodes 1, 2, 5, 7, 10 and 14 are marked as potential neighbors since the configured RSSI threshold is within the blue shaded region that defines the radio range of node 3. Node 3 includes these 7 neighbors in its neighbor list at the end of this state. Similarly, node 9 marks node 6 as a potential neighbor, while nodes 4, 10, 12, 13, 14 become external neighbors to node 9. Nodes 12 and 13 are close to node 4 and get marked as potential neighbors, while node 9 being further away becomes an external neighbor to node 4. Node 11 is out of the RSSI threshold range of node 8 but within the radio range. Thus, node 8 marks node 11 as an external neighbor. The number of neighbors determines the degree of each node, and hence its potential to become a cluster head.

Since the nodes in the network communicate asynchronously, multiple nodes will attempt to transmit during any given time, resulting in collisions. Although the use

of CSMA-CA avoids collisions, it is important that a node is able to complete the transmission without failures or indefinite wait times due to back-off. As discussed in Section 3.2.1.1, to ensure that each node has at least one DeCoRIC message, the transmission time window is computed based on the IEEE 802.15.4 standard [66] including the worst-case back-off. This value is aggregated over the maximum number of nodes supported by the network to form a time window referred to as *round* in DeCoRIC, calculated using the Equation 3.1. The nodes stay in the Discovery state for a *timer* value of one round as shown in Figure 3.4, ensuring that all nodes have successfully transmitted at least one DeCoRIC *ping* message. Based on the CSMA-CA standard discussed in Section 3.2.1.1, the time duration of one round is calculated as:

$$round = N \cdot (\sum_{i=0}^{maxR} \tau_{\text{bo,i}} + \tau_{\text{fr}} + \tau_{\text{ifs}} + +2 \cdot \tau_{\text{cca}}) \tag{3.1}$$

$$\text{where, } \tau_{\text{bo,i}} = (2^{maxBE(i)} - 1) \cdot \tau_{\text{symb}}$$

N is the maximum number of nodes in the network, $maxR$, $\tau_{\text{bo,i}}$ and τ_{cca} are derived from Section 3.2.1.1 representing the maximum retries, worst-case back-off delay at i^{th} re-transmission and clear channel assessment time, respectively. τ_{fr} is the frame transmission time, τ_{ifs} is the minimum inter-frame period, $maxBE$ is the maximum back-off parameter at the i^{th} re-transmission, τ_{symb} is the back-off symbol period and is the clear channel assessment time. The parameters $maxBE$, N and τ_{fr} are configured with the same value at each node (as network parameters). Due to multiple nodes transmitting in parallel, some nodes may not be able to transmit due to interference. Thus, the round is the fundamental time window in DeCoRIC that is configured to be wide enough to accommodate at least one transmission with high probability based on the maximum number of nodes in the network.

Due to the lack of synchronization among nodes, other communication protocols such as Time Division Multiple Access (TDMA) is not used. Additionally, the use of CSMA allows the networks to scale while mechanisms such as TDMA can only accommodate limited slots and require the network to be synchronized. With CSMA, the transmitted frame length is flexible along with the underlying protocol (WiFi, 6LoWPAN, etc.), whereas, while frame length impacts the number of available slots in TDMA.

Election state

In this state, nodes transmit a DeCoRIC message with an up-to-date degree field obtained from the previous state to collectively form a cluster with the neighboring nodes. Each receiving node independently compares its degree to the received degree to keep track of the node with the highest degree (potential CH). Once each node has received at least one transmission from each neighbor (ensured by round configuration) at the end of the discovery state, each node sets itself or its neighbor as the CH based on the highest degree. Thus, every node in the system affiliates itself to a CH, and forms a cluster following the min-cut [67] strategy, with the cut defined by the degree of each node. If multiple nodes have the same highest degree, the node with the lower node

Algorithm 1 Neighbor Discovery state

```
1: LIST: Neighbor, Conn = FALSE
2: Degree = 0, CH.ID = node.ID
3: broadcast(Msg)
4: if rcv() then
5:     Msg = rcv().data
6:     Neighbor[Msg.ID], Conn[Msg.ID] = TRUE
7:     Degree = Degree + 1
8:     if rcv().RSSI < RSSI_threshold then
9:         Neighbor[Msg.ID].ext = TRUE
10:    end if
11: end if
```

Algorithm 2 Cluster Election phase

```
1: broadcast(Msg)
2: if rcv() then
3:     Msg = rcv().data
4:     if Msg.Degree > Degree then  CH.ID = Msg.ID
5:     else if (Msg.Degree == Degree) & (node.ID > Msg.ID) then
6:         CH.ID = Msg.ID
7:     end if
8: end if
```

ID is chosen as CH. This configuration can be altered to choose a higher ID or support priority for specific node IDs. The elected CH oversees communication within its cluster and acts as the gateway for inter-cluster communication. Similar to the discovery state, all nodes keep their radios active during this state and the operation of this state is described in Algorithm 2. Since the CH can be determined with one DeCoRIC message from each node in the network, the clusters can be formed within one round and the nodes transition into the next state at the end of this period. A message from a new node (if any) will be updated into the neighbor list and the connectivity list, while messages from existing nodes reinforce their active state in the connectivity list. If the new node's degree is higher than the current CH, it triggers re-clustering.

From the example system in Figure 3.4, node 3 becomes a CH with nodes 1, 2, 5, 7, 10 and 14 as its members at the end of this state. Node 9 becomes a CH with node 6 as a member, node 4 forms the CH with nodes 12 and 13 as members, while nodes 8 and 11 become independent CHs.

Correction state

Once the clusters are established, there exists the possibility of isolated clusters, i.e., the Cluster Heads are not within each others' radio range, but some common nodes of either cluster can communicate with both CHs and connect the two clusters. To prevent isolated clusters or partitioned networks, each non-CH node verifies the **connectivity** property based on the connectivity list and root-ID fields of the received messages. Any non-CH node satisfying the connectivity property breaks out from the affiliated cluster

Algorithm 3 Cluster Correction state

```
1: broadcast(Msg)
2: if rcv() then
3:     Msg = rcv().data
4:     if (Msg.CHID == Msg.ID) AND (Msg.CHID != CH.ID) then
5:         if Msg.Conn[CH.ID] == 0 then
6:             CH.ID = node.ID   /* Detach from cluster */
7:         end if
8:     end if
9: end if
```

to form a Bridge-CH. This correction process is described in Algorithm 3. If multiple nodes can enable connectivity between the same set of CHs, the rule for CH election is followed, *i.e.*, node with highest ID becomes the Bridge-CH node. In the event of Bridge-CH failure, one of the redundant nodes (bridge) assumes the role of Bridge-CH using the same rule to warrant connectivity. Redundant bridge nodes continue to operate as non-CH nodes, reducing interference to the existing Bridge-CH nodes during inter-cluster communication and, thereby, minimizing their power consumption. The clusters and CHs established are finalized after the Bridge-CH election and the network moves to a stable execution state.

It is energy efficient for the bridge node to break out into a separate CH to facilitate communication between two clusters rather than being a CM of two clusters at the same time. This reduces their radio on time allowing them to be active only during inter-cluster communication. Referring back to the example in Figure 3.4, nodes 10 and 14 identify that they can enable direct connectivity between CH node 9 and their current CH node 3 based on information from the connectivity list. As node 10 has the same degree as node 14, node 10 breaks out as the Bridge-CH because of its lower ID, while node 14 continues as a cluster member. Node 14 is the redundant bridge node that becomes a Bridge-CH in the event of node 10 failure.

The correction state ensures that non-CH nodes strictly remain in low-power mode without involvement in inter-cluster communication while connectivity among nodes is ensured by CH nodes. Further, this state minimizes energy overhead and latency in inter-cluster communication by enabling single-hop connection among CHs, while lowering congestion and error propagation at the Bridge-CH interfaces (hidden node collision problem) [68]. Hidden node collisions are caused due to multiple nodes transmitting to a common node in parallel while the nodes are out of each other's radio range. A node that breaks out from a cluster will not attempt to reintegrate into a cluster and remains an independent cluster head unless a network change invalidates the correction. This ensures that the algorithm converges to an stable state at each node in a time-bound manner without frequent re-clustering.

Stable state

In the Stable state, the CH nodes broadcast a fully populated DeCoRIC frame as *health* message every round. The health message informs the other nodes that the sender node

has not exhausted its energy, allowing every node to detect node failures within its radio range. They also serve as a re-clustering trigger if a change is detected in the network topology. All nodes activate radio duty cycling (RDC) [39] in this state as described in Section 3.2.1.2, keeping the receiver active only in a periodic manner to minimize the power consumed by the radio. Thus, a successful transmission may not guarantee reception at each node. To address this, DeCoRIC defines a configurable period called *cycle* as the minimum set of rounds that will probabilistically ensure that a non-CH node receives at least one transmission from its CH. Cycle duration is computed as:

$$cycle = h \cdot \text{LCM}(txn_{\text{freq}}/h, RDC_{\text{rate}}/h) \tag{3.2}$$

$$\text{where, } h = \text{GCD}(txn_{\text{freq}}, RDC_{\text{rate}}),$$

txn_{freq} is the round duration, RDC_{rate} is the RDC frequency, *i.e,* the number of ON periods per second (number of times the radio is turned on per second), GCD() and LCM() are the greatest common divisor and least common multiple functions, respectively.

Non-CH nodes aggregate the received CH health messages over a cycle and acknowledge them once with a health message at the end of the cycle. The acknowledgment message includes the ID of the leaf node and its own connectivity list (i.e., the list of neighbors from which a *direct* message was received). The per-cycle message from non-CH nodes not only reduces network traffic but also conserves energy at these nodes. The connectivity list aids in failure detection and gets updated upon receiving health messages.

The health messages enable the detection of failed nodes in the system and new nodes in the vicinity. This process is explained further in the following section.

Failure detection. Following the Discovery, Election and Correction states, all the nodes switch to the RDC mechanism and have different transmission intervals depending on their CH/non-CH status. Each non-CH node (or leaf node) only transmits periodic health messages every cycle to ensure that it is still active. Low transmission frequency reduces power consumption for the leaf nodes as well as the channel interference. The CH nodes continue to transmit a health message every round, while the non-CH nodes aggregate the CH health messages every cycle and respond with a health message at the end of a cycle. As the nodes are not synchronized, the sleep windows at each node might be different. Thus, transmissions from a node may be missed by its neighbors, leading to network instability due to false positives about a node's state.

To overcome the above problem, DeCoRIC employs a modified gossiping scheme, derived from [29], to spread information about node health. The gossip mechanism of data transmission is similar to the flooding mechanism without the overhead of significant message transmissions. Each node maintains a *fail* counter (counting rounds) T_{fail} for every node in its neighbor list and it is incremented at every round. Any node which receives a *direct* message from a neighbor node resets the corresponding fail counter (counting rounds) to zero. Further, each node gossips about the health of the neighbor node by including its ID in its connectivity list transmitted during its health message as the nodes within a cluster are in each others' radio range. Meanwhile, if a node receives

Algorithm 4 Stable state

```
1: if rcv() then
2:     Msg = rcv().data
3:     if Msg.Degree >CH.Degree then
4:         CH.ID = Msg.ID
5:         State = ELECTION
6:     end if
7:     for each item i in Conn do
8:         if (Msg.Conn[i] & Conn[i]) then
9:             fail[i] = 0
10:        else if (Msg.Conn[i] & !Conn[i]) then
11:            fail[i] = 0.5 · fail[i]
12:        end if
13:    end for
14:    Neighbor[Msg.ID], Conn[Msg.ID] = TRUE
15: end if
16: if round then
17:    for each item i in Conn do
18:        if fail[i] ≥ T_fail then Conn[i] = FALSE
19:        end if
20:        if fail[i] ≥ 2 · T_fail then Neighbor[i] = FALSE
21:        else fail[i] = fail[i] + 1
22:        end if
23:    end for
24:    if node.ID == CH.ID then broadcast(Msg)
25:    else if cycle then broadcast(Msg)
26:    end if
27: end if
```

a *gossip* message about a neighbor node (i.e., from the connectivity list of a received message), the fail counter corresponding to that node is reduced by half and continues to wait for a direct message from that neighbor as indicated in lines 7-13 of Algorithm 4. When the fail counter corresponding to a neighbor reaches T_{fail} at a node, the neighbor ID is removed from its connectivity list and health message. However, the gossip is only re-initiated (the neighbor ID included in the connectivity list) upon reception of a *direct message* and not based on another gossip message, preventing perpetual gossiping of a failed node.

Assuming the fail-stop fault model, once the fail counter reaches $2 \cdot T_{\text{fail}}$, the corresponding node's ID is marked as failed and also removed from its neighbor list. The stable state operation and the failure detection process are shown in Algorithm 4. The gossiping scheme accommodates false triggers caused by missed packets or transient network conditions, at the expense of increased detection time for node failures. Gossiping also aids in establishing communication and getting updates on its neighbors in an asynchronous network. In our experiments, we observed that the false negatives are virtually non-existent while reliably detecting real failures.

Network adaptation. A failure of a node or the addition of new nodes creates a change in the clustered architecture of the network. The impact of the change ranges from a few clusters (new node addition or non-CH failure) to the entire network (CH failures) depending on the connectivity of the failed node to other clusters. Only the nodes whose degree change is detected as a result of node failure switch to the Election state; unaffected nodes/clusters continue to operate in the Stable state as long as there is no change in the Bridge-CH node. A re-clustering could also be triggered if there are significant network changes across multiple clusters due to dynamic changes in the network (*e.g.* Mobility of nodes, change of Bridge-CH, etc.). Finally, when a new node tries to integrate into the network (i.e., observing health messages), it joins into an existing cluster and could replace the current CH based on its degree that becomes apparent over the next cycle using the *New CH ID* field. If the degree of the new node is higher than CH degree, all the nodes in the cluster undergo re-clustering to establish a new cluster with the new node as the CH. A new node can be integrated into the cluster at run-time by comparing the ID of the received message with its neighbor list irrespective of the state.

In our example network, when node 4 was deleted, it was observed that both nodes 12 and 13 go into the Election state after the detection of the failure. After the Election state, node 12 declares itself as the new CH while node 13 operates as non-CH in the new cluster. Hence in a large network, only the affected cluster elements will switch their state, while unaffected nodes continue in a stable operating state.

3.3 Analysis & evaluation

In this section, we present the evaluation of DeCoRIC using the Cooja Simulator from Contiki [39]. More details on the Cooja simulator can be found in Section 2.4.1. We implemented LEACH and BEEM protocols on Contiki for comparison with DeCoRIC as they are considered as benchmarks in the community. All the three protocols are evaluated with multiple experiments in the same environment. We measure the average power consumption per node of all clustering algorithms over a period of time. Further, as the nodes exhaust their energy, we observe the time for the death (battery exhaustion) of the first node in the network to compare the power efficiency of the protocols. The protocol with the least power consumption and the longest time to death for the first node would have the most power-efficient operation, assuming they offer similar connectivity. We also show the progression of nodes exhausting their energy over time to quantify the power distribution of the protocols among the nodes of the network. This experiment also gives a measure of time during which the network stays intact and connected. To further illustrate the connectivity, we reduce the transmission range of nodes in the simulation to show the time taken and power expended by the protocols to achieve 100% connectivity. Additionally, our test scenarios analyze and evaluate the resilience of DeCoRIC by triggering faults in the network. In this case, we quantify the worst-case delay before the network stabilizes after the re-clustering.

LEACH and BEEM implementation. LEACH [23] and HEED [25] are used as benchmarks in clustering by the community [32]. As BEEM [24] extends HEED ensuring

connectivity, LEACH and BEEM are chosen as representative decentralized techniques for comparison with DeCoRIC. The original LEACH and BEEM implementations were done in MATLAB which abstracts away the low-level communication details (hardware radio model). Due to a common platform, we were able to forgo the radio modeling described in [23]. Also, the MATLAB implementations were inherently centralized since the simulation system has an overall view of the state of each node. Hence, we implemented LEACH and BEEM on Contiki based on the original protocol in MATLAB and the description in the papers [23, 24] [1]. At the lowest level, we use the Contiki radio model as a common platform for emulating LEACH, BEEM and DeCoRIC using the Cooja Simulator; the higher layers are the C implementations of the respective protocols. All the protocol implementations in C are available as open-source for research use.

LEACH and BEEM also perform clustering over a series of states with a steady-state using TDMA among the cluster nodes. The radio of the non-CH nodes on both LEACH and BEEM implementations are turned off once the clustering is complete, except when they have to transmit messages to the CH. Meanwhile, the radio of the CH is always kept on to receive messages from non-CH nodes of the cluster as described by the protocols. We use the synchronization provided by Contiki to achieve TDMA among the nodes for LEACH and BEEM. The TDMA slots are mapped using the RDC mechanism supported by the Contiki radio model.

Both protocols differ in their CH election process which has been implemented according to the description in the respective papers [23, 24]. LEACH uses random probabilistic values to elect CH nodes while BEEM uses degree, residual energy as well as probabilities for the CH election. LEACH does not focus on the connectivity of the network; BEEM uses the areas of high node density that expend more energy to retain energy in other areas extending the connectivity of the network.

Although both protocols differ in their CH election process, they have a cyclic re-clustering mechanism that changes the structure of the underlying clusters completely after a period defined as an *epoch*. An optimal value of the epoch is paramount for energy efficiency on both protocols. We conducted experiments to measure the power consumption by varying different epoch values. On the one hand, it was observed that the time to energy exhaustion decreases and power consumption increases for a smaller epoch since there is a constant re-clustering. On the other hand, higher epoch values yield a lower power consumption and a longer time to energy exhaustion.

Experimental setup. All the experiments were performed on the Cooja simulator with different parameters. DeCoRIC, LEACH and BEEM were run for a group of 50, 100 and 200 nodes arranged in 100 random topologies in an area of 100 x 100 m^2. The topologies are common for all three protocols, with nodes placed at random locations within a given area using the random placement feature of the Cooja simulator. The packet transmission rate is 1 packet/round with a transmission range of 50 m for each node [69]. Round in DeCoRIC, is set based on Equation 3.1 as 0.8, 1.1 and 2.2 seconds, respectively, for 50, 100 and 200 nodes. The lower bound of 0.8 seconds is a restriction imposed by the simulator, below which transmission overlap was observed due to in-

[1] First implementation of a decentralized clustering system with hardware emulation.

Table 3.2: Parameters used in our experimental setup for evaluating DeCoRIC against LEACH and BEEM.

Parameter	Values used
Area (m^2)	100x100
Number of nodes (N)	{50, 100, 200}
Transmission Range (m)	50
CDMA MAC Protocol	CSMA-CA (CXMAC), TDMA
Radio Frequency (GHz)	2.4
Topologies	{Random}
$maxBE$	3
Packet rate (packets/node/round)	1 round
RDC rate (activations/s)	32

complete initialization, resulting in unintended collisions and data loss. Before a frame can complete initialization and start transmission, the timer of the simulator expires to start a new frame initialization, losing the initial frame to be sent. One cycle is configured as 6 rounds, while T_{fail} for CH and T_{fail} for non-CH nodes are computed as 6 and 36 rounds, respectively, using Equations 3.1 and 3.2. To ensure that all three protocols achieve comparable stable state duration before the cyclic re-clustering process, the epoch was chosen to be 10 rounds. As DeCoRIC has transmissions at all nodes at the cycle boundary, the re-clustering interval for LEACH and BEEM was configured to 6 rounds to facilitate equivalent conditions for experiments. This is in congruence with the round configuration chosen for DeCoRIC where transmissions in all nodes occur at the cycle boundary which is configured to be 6 rounds as elaborated in Section 3.2.2, creating equivalent conditions for all the algorithms without any unfair advantage to any of them. The simulation parameters used for our test setup are shown in Table 3.2.

3.3.1 Power consumption

Initially, we evaluate the change in average power consumption of the network and the resulting number of CH nodes by varying the RSSI value in DeCoRIC. Since the CH is primarily responsible for communication with other clusters and maintaining the connectivity within the cluster, a lower number of CHs translates to larger clusters and vice-versa. This trade-off influences the number of messages transmitted externally and is reflected as power savings/expenditure. Figure 3.5 shows the results of the experiment across different RSSI reception thresholds of -45 dBm, -65 dBm and -85 dBm represented by the x-axis for 100 nodes. The y-axis on the left represents the number of CHs formed while the average power consumption per node in milliWatts (mW) is shown on the y-axis to the right. The results show that in the case of -45 dBm nodes (lower effective radio range), less than 30% of nodes act as CH nodes on average with a worst-case of 43% CH nodes. This is a result of many nodes being marked as external neighbors in this case due to their positions in the topology resulting in more than half the nodes acting as CH.

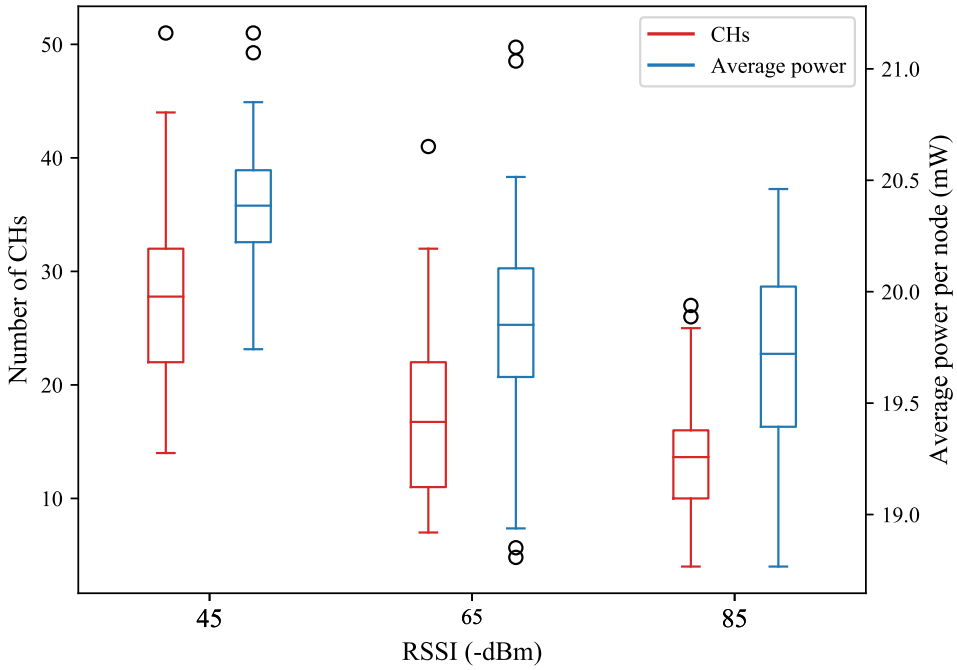


Figure 3.5: Number of CH nodes and average power consumed by the nodes running DeCoRIC over different RSSI thresholds to form clusters.

As the RSSI threshold increases, DeCoRIC marks more nodes as potential neighbors, with a mean and worst case of 13 and 25 CHs at -85 dBm; a mean and worst case of 17 and 33 CHs at -65 dBm. In comparison, most clustering schemes (including LEACH and BEEM) predefine the number of CHs to be between 5–25% (see [45, 47, 34, 49]) of the total number of nodes with no consideration for the network structure, often resulting in disconnected clusters. The outliers in the number of CHs for DeCoRIC can be attributed to the randomness of the topologies since nodes that are farther than the radio range of the RSSI threshold become members of different clusters.

Changing the RSSI results in a change of the cluster size, with higher RSSI leading to bigger clusters and lower RSSI leading to smaller ones. Larger clusters expend higher energy on CHs while reducing the overall network power consumption; smaller clusters result in higher network power consumption with many CH nodes as seen in Figure 3.5. The change in power consumption is more pronounced as the number of nodes scales.

Better clustering in DeCoRIC results in the reduced average power consumption across nodes in the network, as shown by the downward trend in the average power consumption. Further, the outliers in the average power can be attributed to the fact that DeCoRIC employs Bridge-CHs to facilitate connectivity in the network, which increases the average power consumption. The topologies with spatially separated regions of high node density observe a higher number of Bridge-CHs to ensure connectivity between the regions.

The RSSI threshold of -65 dBm is chosen for the rest of the experiments for DeCoRIC. To demonstrate the power saving in DeCoRIC, first, we compare average power per node

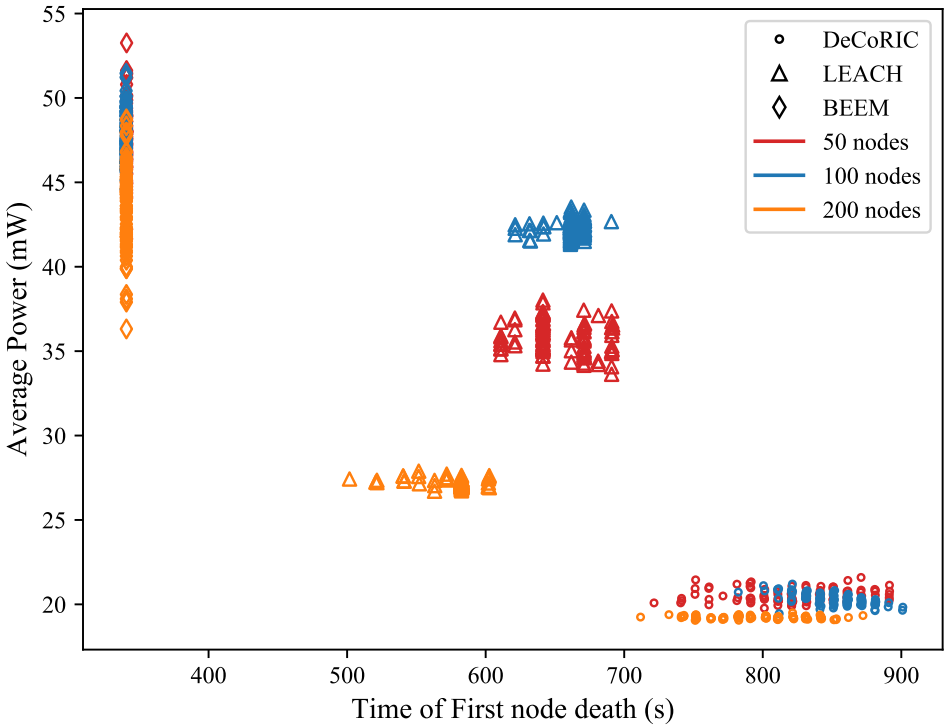


Figure 3.6: Average Power Consumption vs Time of first node death in the network for DeCoRIC compared against LEACH and BEEM.

in milliWatts (mW) along with the time of their first node death (exhaust nodes' power completely) over a simulated duration of 1000 seconds for 50, 100 and 200 nodes in the network across LEACH, BEEM and DeCoRIC.

The results of the first comparison experiment are shown in Figure 3.6, where the x-axis and y-axis represent the time for the first node death in seconds and the average power consumption per node in mW, respectively. The colors represent the number of nodes in the network, while the different shapes represent the protocols. From the results, it is seen that our proposed method has the least power consumption per node, thereby prolonging the time for the first node death. It offers a best-case of 70% and 110% improved power efficiency over LEACH and BEEM for 50 nodes. Similarly, the best-case improvement for the time of first node death is 42% and 109% over LEACH and BEEM for 200 nodes, respectively.

The energy savings in LEACH can be attributed to the proactive load distribution strategy with periodic re-clustering. Hence, the first node exhausts its energy much later compared to BEEM as the power distribution is balanced. Besides, LEACH requires significantly more messages as each node exchanges information about its residual energy to elect the next CH. On the contrary, BEEM adopts a strategy where CH nodes remain unchanged during re-clustering to retain connectivity, while non-CH nodes are retained in a low-power mode during the periodic re-clustering. Due to this strategy, the CH nodes exhaust their power rapidly due to prolonged radio on-time. The periodic clustering

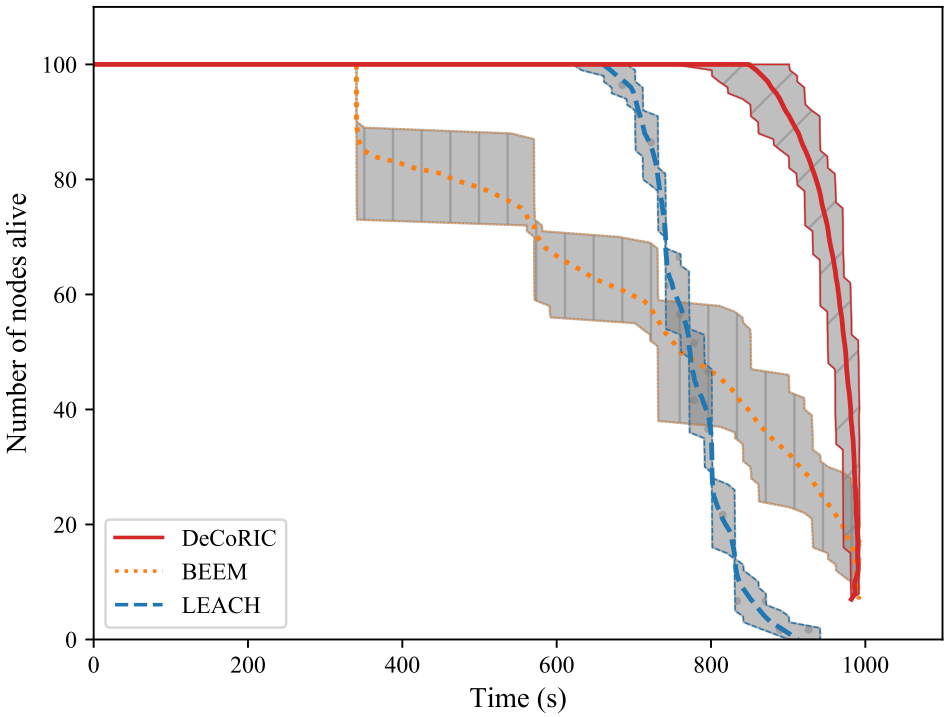


Figure 3.7: Battery drain of the network of DeCoRIC, LEACH and BEEM. The gray area indicates the variation between the minimum and maximum number of alive nodes with the solid line representing the average number of alive nodes.

in LEACH and BEEM forces the nodes to keep their radios on regularly, resulting in higher power consumption for all the nodes. The plot reflects the decrease in total average power as the number of nodes scales while the death of the first node happens faster.

By contrast, DeCoRIC uses a reactive strategy, reducing the activity of the nodes in the Stable state and re-clustering only for node failures. As there is no TDMA, all the nodes experience similar radio activity subject to the density of nodes in the network. Similar to LEACH and BEEM, there is a decrease in power consumption and faster depletion of node power as the network scales. Both BEEM and DeCoRIC retain the CH to ensure connectivity. However, DeCoRIC balances the radio activity efficiently with RDC, re-clustering only if node failures are detected and strives to achieve maximum connectivity. From Figure 3.6, we observe that the power consumption and the active time of nodes are inversely related. The power efficiency of DeCoRIC can be significantly improved with synchronization through lower activity and adapting (re-clustering) only to node/network failures when detected. This makes DeCoRIC ideally suited for critical low-energy IoT systems that need to ensure that the network can adapt to ad-hoc conditions. The synchronization over DeCoRIC will be discussed in Chapter 5.

To maintain connectivity over a longer period, the power dissipation has to be managed efficiently among all the nodes. To quantify the rate of power consumption and the

time of connectivity, we show the time at which the nodes exhaust their energies in a simulation of 1000 seconds. To identify residual energy in the Contiki framework, the power model in [70] was integrated with *Powertrace* to detect the complete drain of energy in the nodes. To begin with, all nodes start with a 2000 μAh (micro Ampere hour) battery capacity. The time at which nodes exhaust their remaining power progressively is shown in Figure 3.7. The y-axis of the plot represents the number of nodes in the network at the start of the simulation while the x-axis represents the time in seconds.

The variation of node lifetime in the three algorithms across different simulations is due to the random placement of nodes in different topologies, resulting in different clusters in each run. The energy exhaustion rate of nodes is higher in LEACH than BEEM and DeCoRIC, as most nodes would have expended similar energies. LEACH has frequent re-clustering where CHs are rotated among different nodes balancing the load of CHs, resulting in an abrupt drop in active nodes when the remaining power is depleted. BEEM has certain nodes in a denser area that start consuming energy after the death of some CH nodes, leading to longer battery life for these nodes. However, since nodes exhaust their energy at an early stage in BEEM, some key bridge nodes could exhaust energy quicker than the other nodes, leading to a disconnected network. DeCoRIC manages power more efficiently with a balanced use of radio and longer sleep times of non-CH nodes using RDC, providing a longer time for the network to stay connected before the nodes exhaust their powers. Since both DeCoRIC and BEEM aim to achieve connectivity, we see that the number of active nodes at the end of the experiment is similar for both algorithms.

3.3.2 Connectivity

We compare the clustering algorithms for their ability to achieve connectivity among the nodes. Connectivity property is paramount in safety-critical applications where data at one part of the network has to be relayed to other parts quickly. In a network of N nodes, there are ${}^{\mathrm{N}}\mathrm{C}_2$ combinations of nodes; some pairs among these establish connectivity across the network through different clusters. CHs and their cluster members form pairs for routing within a cluster while multiple neighboring CHs form a path to route messages across clusters. To test connectivity among nodes of the network, we reduce the transceiver range to 20m, as a larger transmission range in a denser network enables all the nodes to be in communication range with each other. This is a common strategy in dense networks for mitigating collisions, thereby reducing re-transmissions [71]. Reduction in range enables a reduction in transmission power which is the goal for most wireless sensors and IoT devices.

We measure connectivity as the ratio of the number of connected nodes over the total N nodes in the network. Non-CH nodes of a cluster form a connected pair with its CH. Similarly, neighboring CH nodes form a connected path among the clusters. Combining such pairs, we get all the connected nodes in the network (where a routing path exists). Depending on the topology, the maximum connectivity could vary from a single cluster covering a few nodes to multiple clusters covering all N nodes of the network. The former is a result when CHs are not in range of each other, forming independent clusters, and

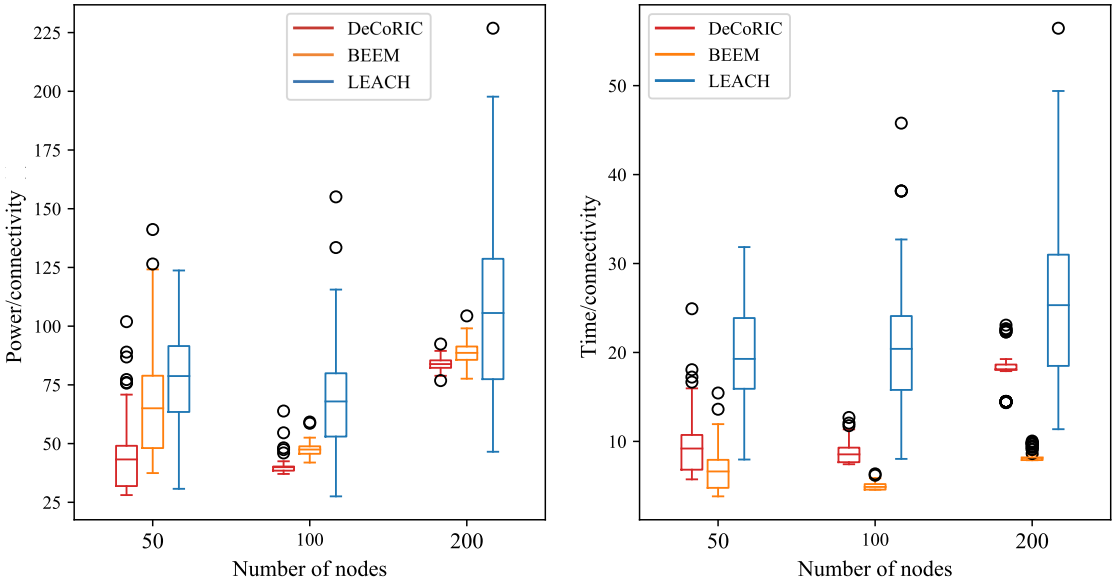


Figure 3.8: Power consumed and time taken for the network clustering normalized by connectivity among all the nodes in the network.

the latter is formed when all CHs are in the range of one another to form a path among all N nodes.

While DeCoRIC and BEEM strive to achieve 100% connectivity, DeCoRIC completes the clustering with less power and a slightly longer time than BEEM. DeCoRIC dynamically determines the number of CHs through node activity and configures RSSI thresholds in a decentralized manner. The Correction state in the DeCoRIC protocol inspects the cluster nodes to check for cases where connectivity can be established. In contrast, LEACH consumes the least time and power for clustering but does not ensure connectivity. Hence, to compare the performance of all the clustering schemes, we normalize both the power (power/connectivity) and time (time/connectivity) in the clustered network by the connectivity achieved by the algorithms.

The results of the comparison are shown in Figure 3.8. The x-axis represents the number of nodes in the network. The y-axis of the left sub-plot represents the ratio of clustering power over connectivity in milliWatts while the y-axis of the right sub-plot indicates the ratio of clustering time over connectivity in seconds. As seen from the left sub-plot, DeCoRIC expends the least power to achieve 100% connectivity, followed by BEEM and LEACH. In contrast to the clustering power, DeCoRIC needs slightly longer to complete clustering compared to BEEM as shown in the right sub-plot. The variations are attributed to the randomness of the topologies yielding different extents of connectivity.

When the size of the network scales up within the same area, the density of nodes increases and consequently, the nodes achieve better connectivity. Although LEACH consumes the least power and time to complete clustering, the results normalized over connectivity (connected nodes/total nodes) show that LEACH would need much higher

time to move towards 100% connectivity. Additionally, with probabilistic CH election and cyclic re-clustering in LEACH, the connectivity varies in the clustered networks over different clustering repetitions. BEEM includes the cyclic re-clustering but retains the same CH to maintain connectivity leading to faster energy exhaustion of these nodes. As we saw in the previous experiment, BEEM has the fastest first node death, leading to a faster loss in connectivity. The 100 randomly sampled topologies prevent any sampling bias and are representative of the changes in connectivity due to the change of CHs resulting from re-clustering changes.

Similarly, the CHs are retained after the clustering is complete in DeCoRIC. However, over multiple epochs, the power consumption reduces significantly for DeCoRIC due to better radio management of the nodes. The longer time of clustering in DeCoRIC is attributed to the Correction state where the number of CH nodes is reduced while striving to attain 100% connectivity. The slightly longer clustering of DeCoRIC creates optimal clusters for better power and node failure management, and hence, sustains the connectivity for a longer time. BEEM is faster as it does not consider the number of CH nodes active while achieving connectivity, thereby expending additional energy and leading to faster node deaths as seen in Figure 3.7. Hence, overall connected time for BEEM is significantly lower than DeCoRIC, where DeCoRIC achieves over 2x longer connected time.

3.3.3 Evaluation of resilience

The round/cycle period in DeCoRIC aims to compensate for the lack of synchronization between the nodes, as explained in Section 3.2.2. Since each round specifies a periodic set of actions (i.e., CH transmission, non-CH nodes receiving without any sequence order), the timing drift between nodes can be bounded to one round. RDC also influences successful message reception as the sleep time of a node may be aligned with the transmission window for the second node. DeCoRIC employs gossiping to overcome this challenge, which allows a node's active state to be propagated by other nodes that receive a direct message. As explained in Section 3.2.2, the failure window (T_{fail}) covers the uncertainties caused due to the asynchronous RDC periods and transmission times, by defining T_{fail} as the least common multiple of the respective time periods.

If a transmission is not received at its neighbor and the neighbor receives a late gossip before its fail counter expires (at T_{fail}), then the counter is halved as it waits to see if it was a transient fault. In an actual system, a health message can be received possibly anywhere within the T_{fail} window at a receiving node. Thus, in the worst-case, there can be an additional half period (of T_{fail}) that a neighbor waits for before declaring the node to have failed. In the case of a node failure, no direct message is received within T_{fail} at receivers, and they stop gossiping about the active state of the node.

To evaluate resilience, we start with a stable network condition that allows us to model the best and worst-case node failures: a Bridge-CH node failure affecting multiple clusters, CH node failure, non-CH node failure and addition of a new node. In each case, we quantified the time taken by the network to detect the change, adapt and restart communication using our simulations. The computed results are shown in Table 3.3. The fail period depends on the activity rate of the node; for CH nodes, their failure

Table 3.3: Best and worst-case reaction time at each node with DeCoRIC in case of network changes.

Network change	**Detection time**		**Recovery time**
	Best-Case	Worst-Case	
Fail: nCH node	$2 \cdot T_{\text{fail}}^{\text{nCH}}$	$2.5 \cdot T_{\text{fail}}^{\text{nCH}}$	immediate
Fail: CH node	$2 \cdot T_{\text{fail}}^{\text{CH}}$	$2.5 \cdot T_{\text{fail}}^{\text{CH}}$	2 rounds
Fail: Bridge-CH node	$2 \cdot T_{\text{fail}}^{\text{CH}}$	$2.5 \cdot T_{\text{fail}}^{\text{CH}}$	2 rounds
Add: New node	3 rounds	1 cycle	[0 or 3] rounds

can be detected within a shorter window ($T_{\text{fail}}^{\text{CH}}$) compared to non-CH nodes since CH nodes transmit more often than non-CH nodes. The constant time taken to detect the failure of a node or the presence of a new node in the radio range is ensured by the activity window approach in DeCoRIC (Section 3.2.2). Once a failure of CH or Bridge-CH node is detected, the nodes switch to the Election state immediately and complete the recovery process over the next 2 rounds. In the case of a non-CH node failure, the recovery is immediate as there are no changes triggered in the cluster itself.

When a new node integrates into the cluster, it can start following a CH within three rounds by listening to its broadcast messages. However, the new node can only determine its degree over the next cycle when other non-CH nodes transmit. If the new node has a higher degree than the current CH, it will transition as the CH by setting the *New CH ID* field of the message frame, causing affiliated nodes to switch to the election state to complete recovery. Otherwise, the recovery is completed immediately, and the new node integrates as a regular non-CH node.

In order to demonstrate the failure detection and update mechanism of DeCoRIC, we measure the time needed to start re-clustering (failure detection) for a CH node and non-CH node failure. During the simulation, a node is randomly deleted from the network during the Stable state. Nodes identify themselves with CH/non-CH status after the clustering is complete. We access the detection time for such random failures in CH (could be a Bridge-CH node) and non-CH nodes. In DeCoRIC, a CH death is detected within $2.5 \cdot T_{\text{fail}}^{\text{CH}}$ and a non-CH death is detected after $2.5 \cdot T_{\text{fail}}^{\text{nCH}}$ including gossiping as described in Table 3.3. We simulated the time for detection of a failure in case of removal of a CH node and a non-CH in DeCoRIC. and the detection time is measured in seconds.

The measured bound was tested and compared with the established theoretical bounds as shown Figure 3.9. As seen in the figure, we observe the detection time in DeCoRIC is in agreement with the theoretical bounds described earlier. As an example for 100 nodes in the network, we observed that the failure was detected at a theoretical bound of 15 rounds in the case of CH failures and 90 rounds in the case of non-CH failures. From Figure 3.9, we notice the worst-case detection was 14 rounds (15.32s) for CH nodes while 75 rounds (81.78s) for non-CH nodes, thus recovering to a stable operating state in a time-bound manner. In case of a non-CH node failure, the stability of the network and the clusters are not impacted even though the detection time is longer than for a

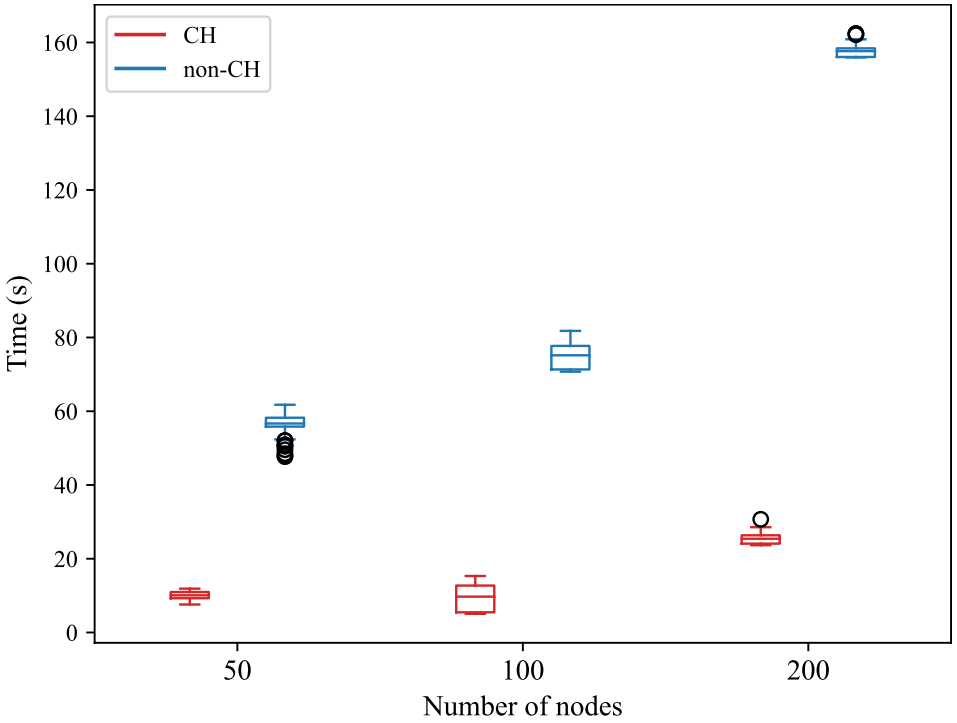


Figure 3.9: Time taken for the network to re-cluster and stabilize in case of a node death.

CH node. Re-clustering occurs if the detected dead/dying node was a CH, triggering a state change to the election state. This operation could be local or extend to a global scale based on the dependencies with other nodes of the network.

Meanwhile, LEACH and BEEM do not have a failure detection mechanism within the protocol. Clustering operation repeats after every epoch, providing a fixed upper bound for the time to re-cluster as there is no explicit failure detection mechanism. Hence, there is no direct comparison between DeCoRIC and these protocols. This property results in a constant recovery time for any CH node independent of the topology changes in the network.

3.4 Summary

In this chapter, we proposed a power-efficient decentralized clustering technique that can dynamically detect and adapt to node failures at runtime while ensuring connectivity among the nodes. The protocol enables the identification of nodes that enable connectivity and strives to create clusters that are connected while consuming minimal power in the process. DeCoRIC provides a resilient and reliable communication framework in a network of any topological structure. We show that the network can re-organize to form new clusters while maintaining connectivity even in case of critical CH failures, with deterministic latency. We implemented the state-of-the-art benchmark clustering protocol

LEACH as well as BEEM, the protocol for connectivity, in the Contiki simulator for comparison with DeCoRIC.

We demonstrate that the number of CHs that are elected independently in DeCoRIC is similar to the number of CHs decided apriori in centralized schemes. We also showed that DeCoRIC achieves up to 70% better power efficiency and 42% longer lifetime compared to LEACH while achieving up to 110% better power efficiency and 109% longer network lifetime in comparison to BEEM. Connectivity is achieved among nodes even in sparse networks using less power by accepting a slightly longer time for the clustering compared to BEEM. DeCoRIC takes the first step towards designing a communication strategy that incorporates all the three requirements of energy efficiency, adaptability and resilience into the IoT network.

4 DeCoRIC use-case: Load Balancing for Mobile devices

In the previous chapter (Chapter 3), we introduced an energy-efficient and resilient clustering scheme DeCoRIC to establish a clustered architecture on which applications can be deployed. To utilize DeCoRIC on a real-world application, we deploy the clustered architecture atop an electric charging infrastructure to achieve demand response in a dynamic load environment. Smart devices and electric vehicles (EVs) form a skewed load amidst existing devices/appliances of the grid due to their variable consumption profiles and geographic mobility.

The current growth in IoT has expanded the communication infrastructure between devices and the grid. IoT-enabled devices can also measure and adjust the energy demand requested from the grid. From the perspective of the grid, scheduling demand requests from all the devices could result in outages and/or prohibitive costs during times when demand exceeds the available peak capacity of the grid [72]. At a 50% higher fuel efficiency than their gasoline counterparts, EVs can consume power of up to 7 average North American households [73, 74]. For a user, the price difference between excess usage and normal usage can be up to a factor of 200 or more [75]. Further, the arrival and the demand of mobile devices such as Electric Vehicles (EVs) is generally not known apriori, leading to uncertainty in demand planning.

At the grid, the penetration of renewables has created *prosumers* (consumers who are producers) who sell their excess energy to other consumers. Renewable sources have alleviated the supply with distributed energy resources (DERs) in the form of aggregators that can supply additional energy in the form of aggregators (micro-and nano-grids) [76]. The additional resources at the supply bridge the disparity between the exponential rise in the number of EVs and the limited charging stations. Traditional demand-response involves devices receiving power from a specific (geographically limited) location of the grid. These solutions aim to ensure that the peak capacity of the aggregator does not exceed while maximizing the number of devices that satisfy their demand at the same geographic location. However, with the availability of multiple aggregators from prosumers, EVs and other mobile devices (EVs being the representative mobile device) communicate with the aggregators and move to different locations to improve the demand-side management (DSM) [33].

The primary goal of DSM is to efficiently utilize the existing grid capacity to meet the demands of the devices without any grid enhancements [77, 78]. Demand satisfaction entails fulfilling the energy demand within a specified time (deadline). Existing works [79, 80] perform DSM by utilizing the day-ahead schedule such that the demand for some devices can be shifted either to an earlier or later time. Additionally, most of the

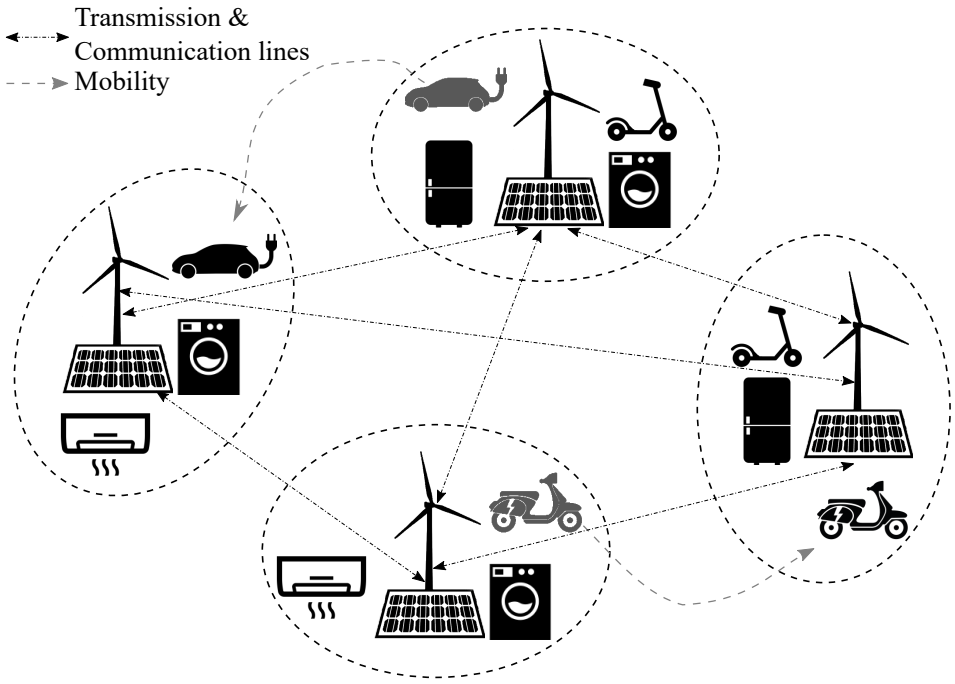


Figure 4.1: The grid aggregator and the devices (including mobile devices formed into clusters)

existing solutions consider only a single mode of operation (unique energy consumption value) for the energy demand of the device during scheduling [33, 77].

Most electrical devices are heterogeneous devices that operate in multiple power modes, *e.g.,* a refrigerator can operate in modes such as defrost, quick freeze, etc. [81]. With mobile devices, the demand can be shifted not only in time but geographically as well. In addition to the demand and deadline parameters, we model the device characteristics such as multiple power modes and mobility. The model's objective is to maximize the utility/benefit of the devices obtained by fulfilling their demanded energy within the deadline.

We consider a two-tier hierarchical network consisting of an aggregator at the higher level (supplier) and devices at the lower level (consumer) as shown in Figure 4.1. Using concepts of connectivity and clustering from the previous chapter, we establish the clustered topology where the CH nodes are the aggregators that manage communication and energy supply. Owing to the geographically-fixed nature of existing grid infrastructure, the aggregators/CH are fixed within a cluster. An aggregator interfaces with the grid and manages the demand from a group of devices within its cluster as CM nodes based on its maximum capacity. Note that this chapter uses the terms CH and aggregators interchangeably. In contrast to DeCoRIC, the aggregators are assumed to be connected to each other using links of a backhaul network similar to existing grids. Devices can turn on/arrive at any time and request their demand to the associated cluster's aggregator. The power supply is realized through an electrical transmission line while the communication is either through a wired or wireless channel.

In this chapter, we develop an efficient scheduling algorithm to maximize the utility of the devices while not exceeding the aggregator's peak capacity at any point. Initially, we demonstrate through a hardware testbed the feasibility of device mobility across different clusters. We consider heterogeneous devices having a different power consumption profile (*e.g.,* EVs, Washing machine, etc.) and various modes of operation (*e.g.,* low-power mode, sleep mode, etc.). Based on the device parameters such as requested demand, deadline, operating modes, etc., the aggregator assigns priorities to devices and schedules them to maximize their utility. To summarize, we make the following contributions:

- Demonstrate device mobility through a simple experiment and establish the foundation for clustered architecture for the load balancing application.
- Generic device modeling: We integrate various device attributes in a model to maximize device utility (benefit).
- Model formulation and solution: We formulate the model and propose an online low complexity heuristic to perform the optimization.
- The model is also implemented in a solver to show the impact of runtime on the solution.
- Experimental verification: We experimentally show the performance of our heuristic for runtime improvement over a solver and a utility loss improvement of over 57.23% over standard scheduling mechanisms such as earliest deadline first, etc. on an unbiased synthetic dataset.

Organization. The rest of the chapter is organized as follows: Section 4.3.1 provides assumptions specific to the use-case of device mobility and charging stations. An experimental study is done to demonstrate the effectiveness of device mobility using decentralized metering in Section 4.1. A discussion on existing state-of-the-art solutions and their shortcomings are made in Section 4.2. Section 4.3 provides the detailed model of the network, devices and the aggregators and translates them into an optimization problem. Sections 4.4 and 4.5 present the proposed heuristic solution and the evaluation of the heuristic solution with a standard solver. Section 4.6 summarizes the contributions of this chapter.

4.1 Device mobility evaluation

A decentralized metering architecture was developed to test the potential of device mobility and current consumption across different geographic locations as a proof of concept. Each device could measure and report its consumption values to its respective CH/aggregator. To ensure data integrity and resilience against faulty data, the reported measurements were stored on a blockchain.

For the demonstration, two Raspberry Pis were used to serve as aggregators for two different clusters [82]. ESP32 Thing boards were used as resource-constrained devices [83]. A current sensor and a battery were connected to one of the ESP32s to measure the

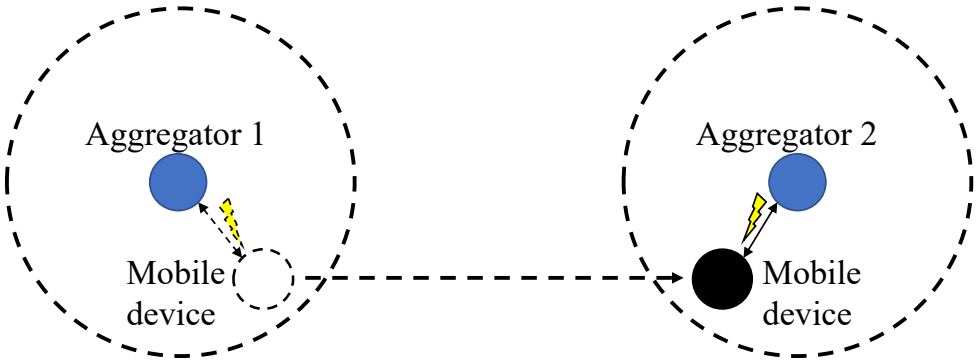


Figure 4.2: Device movement between two clusters and current consumption from different aggregators.

energy consumption and allow mobility (consumption energy for/during movement), respectively [84]. The setup is pictorially represented in Figure **??**.

Membership registration. At the start, devices register themselves with the nearest cluster as a member using the membership request. As a response, the aggregator sends its address which the device registers itself as a membership ID and starts transmitting the consumption data. In the event of a change in the cluster due to mobility, the consumption data is temporarily stored locally within the device before registering with a new cluster. The new aggregator gets updated for sending the consumption information. Furthermore, a device registration with the first cluster can be retained as ownership of the device with the first cluster. The ownership allows the aggregator of the second cluster to transmit the information back to the first cluster using the backhaul network. The messages exchanged during the membership and mobility are shown in Figure 4.3.

Mobility test. The above process is tested with a simple network of two clusters and allowing a mobile device to migrate from cluster 1 to cluster 2. This experiment is hence designed to validate the claim that the device consumption can be monitored even when it is operated at different grid locations. The consumption data of the device during the network transition obtained from Aggregator 1 is shown in Figure 4.4. The device is initially registered with Cluster 1 and reports its consumption value to Aggregator 1. The reported values (until it gets disconnected from Cluster 1) are shown in the left half of the figure. The pre-configured measurement interval for the device, T_{measure}, was set to 10 times per second i.e., the device consumption is reported to the aggregator every 100 milliseconds. The time interval is achieved using the RDC feature of the device (RDC_{rate}) discussed in the previous chapter. If the device is disconnected before the reporting time, the data is stored locally until the network is restored. As discussed earlier, the consumption during mobility is feasible due to the availability of battery power within the devices. After a short wait time of an hour, the device is moved from Cluster 1 to Cluster 2.

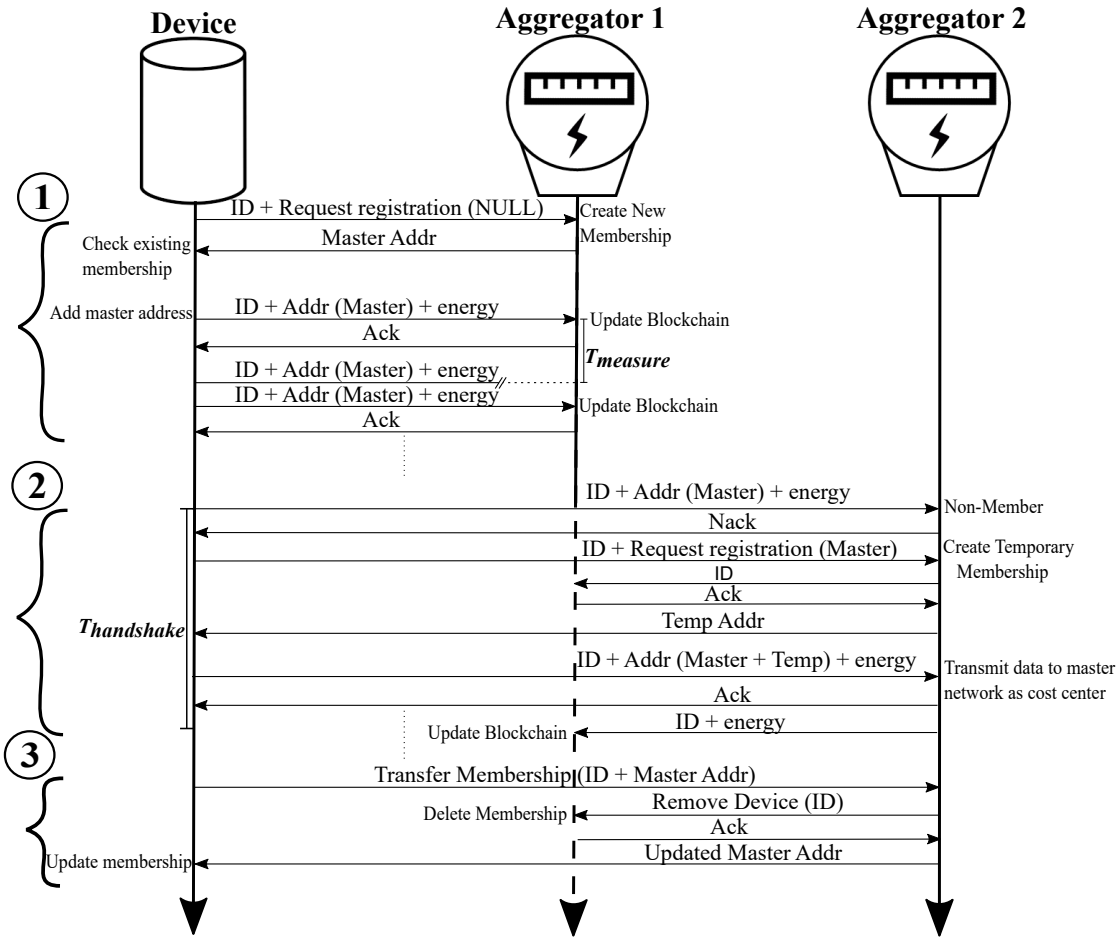


Figure 4.3: Device registration and handshake messages for mobility between two clusters.

When the device is moving across different clusters the power is derived from the battery with no consumption from the grid (as it is not connected to the transmission line) and this duration is denoted as *Idle* time in the figure. Once the device establishes an electrical connection with a different aggregator cluster, it continuously scans the cluster messages to determine its reporting aggregator address (Aggregator 2 in our case). The device stores its consumption (marked by the blue line in the figure) where the logging is rerouted to the onboard storage during the handshake process to local storage. The energy consumed from the battery will be added to the energy demand of the device that will be fulfilled by the aggregator. After establishing a connection with the new cluster (*i.e.,* temporary membership registration), the device transmits consumption data and any locally stored data to Aggregator 2. Additionally, based on the previous membership/ownership of the device, Cluster 1 is informed of the consumed energy using the transmission link between clusters 1 and 2, allowing consolidated billing for the device in Cluster 1. The time to register a temporary membership in Cluster 2, $T_{\text{handshake}}$, is found to be 6 seconds on average with a variation between $5.5 - 6.5$ seconds

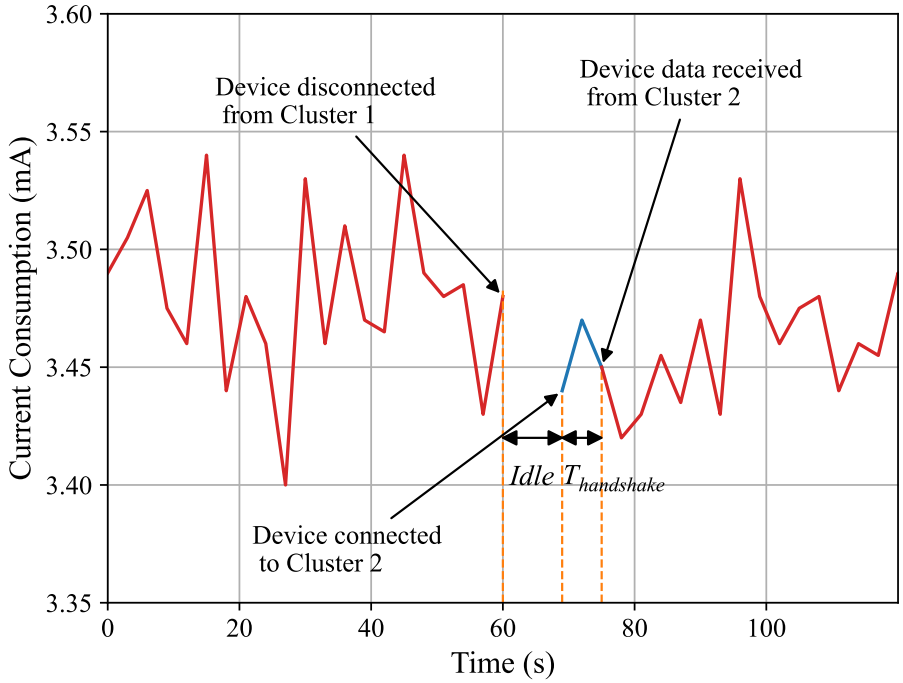


Figure 4.4: Device transition of current consumption and reporting after moving from cluster 1 to 2.

over 15 runs. The data communication between aggregators does not incur much delay (1 millisecond) as the backhaul network is assumed to have high bandwidth.

4.2 Related Work

Load scheduling for DSM in electric grids is a well-studied problem in literature [77]. The existing works in DSM can be categorized broadly based on devices being grid-powered and devices being both grid and battery-powered. The former analyzes the energy demand from stationary devices, while the latter addresses the demand from battery-powered mobile devices such as EVs.

Various solutions in DSM exploit load flexibility to shift the demand to a later time to minimize the peak consumption at any given time [79]. Yu and Hong [85] aim to balance the demand and supply by formulating the interaction between the aggregator and devices as a Stackelberg game where both entities maximize their utility until an equilibrium is reached. Noor et. al. [86] introduced blockchain to the game-theoretic framework for a trustless peer-to-peer consensus mechanism between aggregator and the devices. Zhao et. al. [87] proposed augmenting central energy storage to the grid and virtually distributing its capacity to the devices at different time steps to maximize the operational profit. Kim and Dvorkin [88] extended the concept of [87] with mobile energy storage that provisions for additional capacity in emergencies such as natural disaster services to minimize the investment and operational cost. To overcome the

computational issue of game theory-based solutions, Newton method [89] and proximal decomposition algorithm [90] were used to accelerate the run-time convergence to the Nash equilibrium, and to find the optimal EV schedule that reduces the gap between peak load to average load. Roh and Lee [91] categorize and model devices based on the flexibility of their preemption, scheduling and causality to maximize their utility using generalized bender's decomposition. Based on a similar categorization of load and to minimize peak load, a water-filling algorithm was used to provide an exact solution [92]. Adika and Wang[20] designed a scheduling mechanism using linear programming (LP) for grid-powered and battery-storage devices to exploit charging and discharging in off-peak and peak times respectively, to minimize the consumption cost. Chiu et.al. [93] not only maximize device utility but also minimize the consumption cost and carbon emissions at the grid in a multi-objective formulation and provide a distributed solution using Lagrange decomposition. Restructuring the smart grid network into multiple layers for faster bring-up of the network in case of failure [17] and estimating the grid parameters to compare and detect faulty information are some of the measures taken by works focused on network resilience [28]. Meta-heuristics such as genetic algorithm [94] etc. are utilized to model and maximize device utility and obtain a schedule for load-shifting. Although the above works minimize the aggregators' peak load, devices are restricted to only a single mode of operation and do not account for the stochastic arrival of mobile devices. Practically, devices have multiple modes of operation that can be exploited to serve power to more devices within their deadlines [95].

Other works in the literature focus on EV scheduling to minimize the peak load on the aggregators. In contrast to grids where good estimates of daily, weekly or monthly schedules are available, there is more uncertainty with mobile devices such as EVs due to their stochastic arrivals and demands. The authors of [96] alleviate this issue and minimize the consumption cost by letting devices estimate the day-ahead load and play a game with aggregators to minimize the deviation between actual load and estimated load. Other methods adjust the generation of renewable resources depending on the demand from EVs [97, 80]. Zhang and Cai [80] focus on increasing the profit for aggregators by utilizing Model Predictive Control (MPC) to estimate renewable generation to adjust the EV schedule whereas, Schuller et.al. [97] focus on reducing reliance on the grid using empirical EV data and LP to maximize renewable utilization and optimal EV scheduling. Zheng et.al. [98] propose a distributed solution at the aggregator-level using MPC and fuzzy logic to compute the optimal EV schedule to minimize the computational overhead of centralized solutions. Similar to grid-powered devices, a Stackelberg game was used between aggregators and EVs to set prices proportional to the demand to maximize the aggregator utilization and hence, the number of EVs served [72]. However, the authors only use price-based arrival probabilities for EVs and do not track the EVs' arrival at different aggregators. Game theory was also used to mitigate anomalous EVs consuming the bulk of power and aim to achieve a minimum utility for any EV by committing an energy budget upon arrival and re-distributing the remaining energy budget to decide on the admittance of all the new incoming EVs [99]. A priority function is introduced based on the demand and deadline of the EVs and formulated as a binary optimization problem by relaxing the standard form of linear optimization [100]. Rassaei et. al. [101]

consider stochastic static loads (non-mobile) as well as EVs (mobile) to minimize the cost of consumption for devices and peak load for aggregator through a decentralized game-theoretic solution and a centralized interior-point solution, respectively. A probabilistic threshold was used for price comparison to decide whether to provide energy to an EV for unknown pricing while using a combinatorial search method for known pricing [102]. Zhu et.al. [7] formulate a multi-stage optimization with EV scheduling followed by fixed device scheduling consider the possibility of changing the EV aggregator at arrival time if the aggregator is fully loaded. Although they consider multiple modes of charging, they are linked to the time of the day rather than instantaneous demand. Based on the fact that price information is not available for EVs to schedule their charging in real-time, Yi et. al. [103] propose deterministic online algorithms based on the charging rate and prove a bounded performance. They also provide an optimal offline algorithm when the pricing information is known. The above works on EVs are focused on optimizations to obtain EV schedules assuming EVs cannot change their geographical location. EVs can move to different geographical locations to fulfill their energy demand if there is time-bound or a lack of supply. However, by exploiting the mobility property of the EVs, movement across different aggregators can alleviate the peak load and achieve higher utility for EVs. Load scheduling of heterogeneous devices (mobile and non-mobile) with different energy consumption profiles is the novelty of our proposed solution.

To the best of our knowledge, no work in the literature exploits device mobility while considering multiple modes of operation of devices.

4.3 System model

In this section, we introduce the model formulation and the objective function. The important notations used in this chapter are listed in Table 4.1.

4.3.1 Use-case-specific assumptions

We make the following assumptions for formulating our model and objective:

- The network is reliable and trustworthy, i.e., the underlying communication framework from DeCoRIC prevents message loss or tampered data.
- Each device is connected only to one aggregator at any time
- For every device d_k every available power mode (except $\alpha_{k,0} = 0$) and the total requested energy are positive integer multiples of $\alpha_{k,1}$, i.e., $\forall \alpha_{k,z} \in \alpha_k \setminus \{0\}, E_k + I_k = E_k^{total}, n \in \mathbb{N} : \alpha_{k,z} = n\alpha_{k,1}, E_k^{total} = n\alpha_{k,1}$
- The devices d_k^i of cluster i can directly communicate to other devices within the same cluster but cannot communicate to devices d_k^j of a different cluster j.
- Serving a portion of the total energy E_k^d of a device increases its utility by the same proportion since it is normalized by E_k^{total}.

Table 4.1: Notations and associated description used for modeling.

Notation	Description
Network parameters:	
$\mathcal{A} = \{a_{1\ldots j\ldots J}\}$	Set of J aggregators
$\hat{\alpha_j}$	Power capacity of aggregator a_j
$c_{a_j \to a_{\hat{j}}}$	Cost per unit time for movement from a_j to $a_{\hat{j}}$
$\delta_{a_j \to a_{\hat{j}}}$	Time for movement from a_j to $a_{\hat{j}}$
$\mathcal{C} = \{c_{a_j \to a_{\hat{j}}}, \ldots\}$	Set of all movement costs across all aggregators
$\mathcal{T}$	Time horizon with τ slots indexed by t
T_0	Length of one time slot
τ	Number of time slots
Device parameters:	
$\mathcal{D} = \{d_{1\ldots k\ldots K}\}$	Set of K devices
$\{d_{k,j}\}$	Set of devices in the cluster associated to aggregator a_j
E_k	Energy demanded of d_k
I_k	Initial energy of d_k
T_k	Deadline of d_k
R_k	Arrival/Release time of d_k
m_k	Binary variable denoting if d_k is mobile
$\alpha_k = \{\alpha_{k,0} \ldots \alpha_{k,i} \ldots \alpha_{k,n}\}$	Set of n charging modes of d_k
κ_k	Criticality of d_k indicating the utility loss rate
$\gamma_{mk}[a_j, a_{\hat{j}}, t]$	Decision variable if device d_k moves from a_j to $a_{\hat{j}}$
$\gamma_{pk}[i, a_j, t]$	Decision variable if device d_k receives its ith power mode from aggregator a_j
$\mathcal{P}_k(t)$	Total accumulated utility up to t for d_k
$p_k(t)$	Utility achieved at t for d_k

4.3.2 Network-level parameters

The network architecture comprises J aggregators, each with its own cluster and K devices connected to each other as shown in Figure 4.1. All the devices in the network are in the set $\mathcal{D} = \{d_1, d_2, \ldots, d_k, \ldots d_K\}$ and all aggregators are represented in the set $\mathcal{A} = \{a_1, a_2, \ldots, a_j, \ldots a_L\}$. A device d_k and aggregator a_j are represented by their unique IDs k and j, respectively. Every device in $d_k \in \mathcal{D}$ receives power from one of the aggregators $a_j \in \mathcal{A}$ by communicating information such as demand, deadline, available power at the aggregator, etc. $\{d_{k,j}\}$ denotes the set of all devices (including new/mobile device arrivals) in the cluster associated to a_j.

Every aggregator a_j has a fixed power budget $\hat{\alpha_j}$ used for serving power demands of devices $\{d_{k,j}\}$. Every aggregator a_j is assumed to be connected to every other aggregator in set $\mathcal{A}$ using a backhaul network.

4.3.3 Device-level parameters

Devices request energy with a timing constraint/deadline (*e.g.,* T_k for d_k) to fulfill their demand. They are heterogeneous with each device having a different power consumption profile (*e.g.,* EVs, Washing machine, etc.) and various modes of operation (*e.g.,* low-power mode, sleep mode, etc.). Each device makes use of one of its power modes at every time instant to fulfill its energy demand of E_k. Few devices have an additional property of mobility where they can move across different clusters if power is unavailable at the aggregator of the incumbent cluster. Each device d_k has a deadline T_k associated with its task before which it has to fulfill its energy demand. Devices that have sufficient time before their deadline to receive power can be treated as flexible loads since they can receive power at a later time, increasing the elasticity on the demand side. Hence, a device request θ_k is represented as:

$$\theta_k = (R_k, T_k, m_k, I_k, E_k, \kappa_k, \alpha_k) \tag{4.1}$$

where R_k denotes the arrival time slot when the device requests power. The arrival slot implies that either a new device switches on at R_k or an existing device moves from one aggregator and arrives at the new aggregator at R_k. I_k is the initial energy available with the device at the time of arrival (e.g. a battery's State-of-Charge). The total energy E_k^{total} is the total energy capacity of a device which is the sum of its initial charge I_k and the demanded energy E_k. The positive constant κ_k is used to implement different criticality among devices, e.g., emergency light (high κ) vs reading light (medium κ) vs washing machine (low κ). κ_k also ensures that devices with higher demand do not always translate to a higher priority. We assume that the devices' energy demand is feasible within T_k with one of its power modes, i.e., $E_k \leq p_k(t) \cdot (T_k - R_k)$ where $p_k(t)$ is the chosen power mode at time t from the set α_k, *i.e.,* $p_k(t) \in \{\alpha k\}$. m_k denotes whether the device is mobile ($m_k = 1$), or non-mobile ($m_k = 0$).

Each device d_k has a power mode $\alpha_{k,0} = 0$ when the device is not served any power, as well as up to $n \in \mathcal{N}$ power modes in increasing order of consumption, i.e., $\alpha_{k,0} < \alpha_{k,1} < \ldots < \alpha_{k,n}$. The number of power modes and the values for each mode are

specific to each device. Hence, the set of all power modes of device d_k is given as $\alpha_k = \{\alpha_{k,0}, \alpha_{k,1}, \ldots, \alpha_{k,n}\}$.

Mobility. Mobile devices can move across different clusters as they are equipped with energy storage such as batteries that facilitate energy required for movement. Any device d_k that moves from its current cluster with the aggregator a_j to another (target) cluster with the aggregator $a_{\hat{j}}$ has a delay $\delta_{a_j \to a_{\hat{j}}}$ and an associated cost $c_{a_j \to a_{\hat{j}}}$ per unit delay to pay for the movement represented by the tuple $< \delta_{a_j \to a_{\hat{j}}}, c_{a_j \to a_{\hat{j}}} >$. The delay $\delta_{a_j \to a_{\hat{j}}}$ from current time t_c is the arrival time of d_k at aggregator $a_{\hat{j}}$, i.e., R_k of d_k at $a_{\hat{j}}$ is $t_c + \delta_{a_j \to a_{\hat{j}}}$. Movement delays are representative of the distance of the aggregators from each other and the movement costs are proportional to the distance. E.g., an EV that moves for a longer distance consumes higher cost and incurs a higher delay and vice-versa. The movement option for any mobile device to move between any two aggregators j and $\hat{j}$ are given by,

$$\mathcal{C}_{a_j \to a_{\hat{j}}} = \{< \delta_{a_j \to a_{\hat{j}}}, c_{a_j \to a_{\hat{j}}} > \ldots\ldots\} \tag{4.2}$$

Depending on the movement option chosen, the cost and the time vary. The total cost (energy needed) of a movement combination is given by the product, i.e., $c_{a_j \to a_{\hat{j}}} = \delta_{a_j \to a_{\hat{j}}} \cdot c_{a_j \to a_{\hat{j}}}$. The demanded energy E_k is the sum of the demand requested in θ_k and the energy consumed during the movement of the device $c_{a_j \to a_{\hat{j}}}$.

Without loss of generality, movement times are ordered in increasing order of movement time. Since the above tuple specifies the movement options only between two clusters, a matrix $\mathcal{C}$ is defined to include all movement options across any two clusters a_j and $a_{\hat{j}}$ in the network and is given by:

$$\mathcal{C} = \begin{bmatrix} 0 & \ldots & (1 \to \hat{j}) \\ (2 \to 1) & 0 & \vdots \\ \vdots & \ddots & (j-1 \to \hat{j}) \\ (j \to 1) & \ldots & 0 \end{bmatrix} \tag{4.3}$$

The number of available movement options and hence, the costs, may differ across different clusters ($a_j, a_{\hat{j}}$ pairs). Mobile devices are limited by their initial/available energy while choosing the movement options, i.e., the movement cost cannot be greater than their available energy. However, the chosen movement cost is an additional energy derived from the aggregator along with its demand E_k.

4.3.4 Utility function

Assuming a discrete-time system, the total time $\mathcal{T}$ is divided into τ time slots of length T_0 and denoted as $\mathcal{T} = \{1, \ldots, \tau\}$. Each device achieves a certain utility (benefit) when it is served power at any time slot. A utility $p_k(t)$ is achieved by a device at any given time slot t for a duration T_0 when it is served with power $\alpha_{k,i}$ that is among the power modes α_k *i.e.,* $p_k(t) = \alpha_{k,i}$ at time t. The utility $p_k(t)$ of each time slot is appended to

the set of accumulated utility $\{p_k(0), p_k(1), \ldots, p_k(t)\}$. Hence, at the current time slot t_c, $\mathcal{P}_k(t_c)$ is the sum over the utility $p_k(t)$ of all the previous time steps leading to t_c.

$$\mathcal{P}_k(t_c) = \sum_{t=0}^{t_c} p_k(t)\gamma_{pk}[i, a_j, t] \tag{4.4}$$

where $\gamma_{pk}[i, a_j, t]$ indicates that ith power mode was served at cluster with aggregator a_j at time t. In an ideal case, all devices are scheduled before their T_k and served with the corresponding energy demand E_k, yielding a utility of E_k without any loss. However, in practice, due to congestion and devices with higher demand, scheduling all devices without deadline misses or movement is not feasible. Devices incur a utility loss when it moves (as described earlier) or misses its deadline that is time-dependent. The total utility loss function for devices per time slot t is given by:

$$\begin{aligned} u_k(t) = \beta_k^d(t) + 2 \cdot \beta_k^m(t) \\ + (1 - m_k)(1 - \delta_{a_j \to a_j}(t))\beta^{max} \end{aligned} \tag{4.5}$$

The utility loss function in Equation (4.5) consists of three terms: loss due to mobility, loss for missing the deadline and penalty for moving stationary devices across clusters.

Utility loss due to deadline violation

The first term in Equation (4.5) incorporates the penalty for missing the deadline. It is given as

$$\beta_k^d(t) = \begin{cases} f(t), & (t > T_k) \wedge (\mathcal{P}_k(t) \leq E_k) \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

where $f(t)$ is given by $(\mathcal{P}_k(t_c) - E_k) \cdot [\exp(\kappa_k(t_c - T_k))]$. $\mathcal{P}_k(t)$ is the accumulated utility (consumed energy) up to the current time slot t_c as defined in Equation (4.4). The expression in Equation (4.6) is the loss factor per time slot equivalent to $\alpha_{k,n}$ after the deadline is missed. κ_k provides a measure of criticality in devices to signify how imperative it is to serve a device d_k to minimize the significant losses in utility. A higher value for κ_k translates to a higher rate of loss for exceeding the deadline.

Permanent utility loss due to mobility

Mobile devices consume additional energy than their initial demand to finish their tasks owing to energy loss during movement. This loss in energy translates to utility loss. In order to compensate the additional energy for mobility, the total demand of the device is updated to include the energy for mobility, and consequently, increases the energy consumption from the aggregator. This additional energy (new energy demand) supplied by the aggregator due to mobility on top of the initial energy demand of all the devices leads to a utility loss on the grid which cannot be compensated. Hence, there

is a factor of two for the second term for utility loss due to mobility. The loss due to mobility is given as:

$$\beta_k^m(t) = c_{a_j \to a_{\hat{j}}} \cdot \gamma_{mk}[a_j, a_{\hat{j}}, t] \tag{4.7}$$

where, $c_{a_j \to a_{\hat{j}}}(t)$ is the cost per time slot derived from Equation (4.3) and $\gamma_{mk}[a_j, a_{\hat{j}}, t]$ represents the binary variable that indicates if a device is chosen to move (= 1) or not (= 0) between clusters with aggregators a_j and $a_{\hat{j}}$ at time t. It is important to note that if a mobile device with an initial charge moves to another cluster before consuming power, it results in a negative utility that is capped at $-I_k$ as it is the maximum supply from the battery.

Utility loss for moving stationary devices

The third term in Equation (4.5) prevents the stationary devices to move from their parent cluster by imposing a very high penalty on those devices. If a device d_k belongs to a cluster a_j and it moves to a different cluster $a_{\hat{j}}$, then the penalty is given by

$$(1 - m_k)(1 - \delta_{a_j \to a_{\hat{j}}})\beta^{max} \tag{4.8}$$

with $\delta_{a_j \to a_{\hat{j}}}$ is 0 if $(a_j = a_{\hat{j}})$, and 1 otherwise. β^{max} is the maximum penalty constant configured to a very high value.

4.3.5 Objective function and constraints

Given all devices $d_k \in \mathcal{D}$ and the utility function in Equation (4.5), the objective is to minimize the cumulative utility loss across all devices over all the time slots in the system. It is important to note that the objective is to allow a device with high criticality to consume power at the earliest to reduce the penalty in utility rather than maximizing the devices to meet their deadlines. Hence, there could be a higher number of deadline violations while minimizing the utility loss.

Since the progress $\mathcal{P}_k(t)$ remains the same independent of the mobility or the time of receiving power, we can translate the maximization of utility into a minimization of utility loss. As the stationary devices do not move, the utility loss term is given by the combination of terms one and two in Equation (4.5), i.e., $u_k(t) = 2 \cdot \beta_k^m(t) + \beta_k^d(t)$. The decision variables are the choice of each device d_k to receive power (γ_{pk}) and the choice to move to another cluster (γ_{mk}).

The objective function and the constraints are given as:

$$\min \sum_{t \in \mathcal{T}} \sum_{d_k \in D} u_k(t) \tag{4.9}$$

$$\text{s.t.} \sum_{j \in A} \sum_{i \in \alpha_k} \gamma_{pk}[i, a_j, t] \leq 1, \qquad \forall t,\ \forall d_k \in \mathcal{D} \tag{i}$$

$$\mathrm{c}_{a_j \to a_{\hat{j}}} \in \mathcal{C}, \qquad \forall t,\ \forall \{a_j, a_{\hat{j}}\} \in A \tag{ii}$$

$$\sum_{a_j \in A} \sum_{a_{\hat{j}} \in \mathcal{A}} \gamma_{mk}[a_j, a_{\hat{j}}, k] = 1, \qquad \forall t,\ \forall d_k \in \mathcal{D} \tag{iii}$$

$$\sum_{i \in \alpha_k} ((1 - m_k) \cdot p_k(t) \cdot \gamma_{pk}[i, a_j, t] + m_k \cdot p_k(t) \cdot \gamma_{pk}[i, a_j, t] \cdot \gamma_{mk}[a_j, a_{\hat{j}}, t]) \leq \hat{\alpha_j}, \forall t,\ \forall \{a_j, a_{\hat{j}}\} \in \mathcal{A},\ \forall d_k \in \mathcal{D} \tag{iv}$$

$$\gamma_{mk}[a_j, a_{\hat{j}}, t] + \gamma_{mk}[a_{\hat{j}}, a_{\hat{j}}, t] - \gamma_{mk}[a_j, a_{\hat{j}}, t-1] \geq 0, \quad \forall d_k \in \mathcal{D},\ \forall \{a_j, a_{\hat{j}}\} \in \mathcal{A},\ \forall t,\ \forall a_j! = a_{\hat{j}} \tag{v}$$

$$\gamma_{mk}[a_j, a_{\hat{j}}, t] + \sum_{a_{\hat{j}} \in \mathcal{A}} \gamma_{mk}[a_j, a_{\hat{j}}, t] - \gamma_{mk}[a_j, a_j, t-1] \geq 0, \quad \forall d_k \in \mathcal{D},\ \forall \{a_j, a_{\hat{j}}\} \in \mathcal{A},\ \forall t,\ \forall a_j! = a_{\hat{j}} \tag{vi}$$

$$\gamma_{mk}[a_j, a_{\hat{j}}, t] \cdot \delta_{a_j \to a_{\hat{j}}} + \Bigg(\gamma_{mk}[a_j, a_{\hat{j}}, t-1] \cdot \Big(\sum_{t_m=1}^{\delta_{a_j \to a_{\hat{j}}}} (\gamma_{m,k}[a_j, a_{\hat{j}}, t - t_0]) - \delta_{a_j \to a_{\hat{j}}} \Big) \Bigg) \geq 0, \quad \forall d_k \in \mathcal{D},\ \forall \{a_j, a_{\hat{j}}\} \in \mathcal{A},\ \forall t,\ \forall a_j! = a_{\hat{j}} \tag{vii}$$

$$\sum_{t_m=1}^{\delta_{a_j \to a_{\hat{j}}}} \gamma_{mk}[a_j, a_{\hat{j}}, t - t_0] \leq 0, \quad \forall d_k \in \mathcal{D},\ \forall \{a_j, a_{\hat{j}}\} \in \mathcal{A},\ \forall t,\ \forall a_j! = a_{\hat{j}} \tag{viii}$$

$$\sum_{t \in T} (1 - m_k) \cdot p_k(t) \cdot \gamma_{pk}[i, a_j, t] + m_k \cdot p_k(t) \cdot \gamma_{pk}[i, a_j, t] \cdot \gamma_{mk}[a_j, a_{\hat{j}}, t] \leq E_k,\ \forall d_k \in \mathcal{D} \tag{ix}$$

where Constraints (i) and (ii) ensure that the power mode chosen in any time slot is from the set of available power modes at only one of the aggregators and movement options, respectively. The state of a device can either be stationary or in motion as indicated by Constraint (iii). Constraint (iv) restricts the power used by all devices of a cluster to not exceed the maximum power budget $\hat{\alpha_j}$ of its aggregator a_j for all time slots. Constraints (v) and (vi) shows the behavioral constraint on the state of the device in the current time slot based on the state in the previous time slot, *i.e,* a device has a mutually exclusive state of being stationary or moving and a device cannot consume power during movement. The movement of a device for at least $\delta_{a_j \to a_{\hat{j}}}$ between clusters with aggregators a_j and $a_{\hat{j}}$ is specified as a constraint in Constraints (vii) and (viii). The total accumulated energy of a device across all time slots is bounded by E_k as shown

by Constraint (ix). The utility loss due to mobility results in an updated (increased) energy demand to compensate for the energy spent on movement.

The objective function in Equation (4.9) has integer constraints and binary decision variables γ_{mk} and γ_{pk} with the choice of power modes of a device and is formulated for a discrete-time. Hence, the problem is a mixed-integer non-linear programming problem (MINLP). For this problem, we try to serve the device with the best utility within the constrained power budget of the aggregator. The optimization in Equation (4.9) with its constraints is an NP-hard problem due to the combinatorial nature of picking one of many power modes and aggregators and its similarity to multiple-choice multiple-knapsack problem [104] which is known to be NP-hard.

Theorem 1. *The optimization in Equation (4.9) with constraints is an NP-hard problem.*

Proof. In a typical 0-1 knapsack problem, items are picked from an available list to maximize the value of the items under a constrained weight. If there are multiple options for a single item available as a set to uniquely pick one item (*e.g.*, different colors of the same item), the problem is called a multiple-choice knapsack problem. With multiple such knapsacks, the problem is transformed as a multiple-choice multi-knapsack problem (MCMKP) [105] which is proven to be NP-hard. Formally, MCMKP problem can be expressed as:

$$max. \sum_{i=1}^{m} \sum_{j=1}^{n_i} p_{ij} x_{ij} \tag{4.10}$$

$$\text{s.t.} \sum_{i=1}^{m} \sum_{j=1}^{n_i} w_{ijk} x_{ij} \leq c_k, \qquad k = 1, \ldots, l \tag{4.11}$$

$$\sum_{j=1}^{n_i} x_{ij} = 2, \qquad i = 1, \ldots, m \tag{4.12}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, m, j = 1, \ldots, n_i \tag{4.13}$$

Equation 4.10 is the objective function where p_{ij} is the profit to be maximized by picking j items among n_i of i different classes among m class of items. w_{ijk} indicates the k-dimension weight among l dimensions while x_{ij} indicates the binary decision of picking an item under the constraints listed in Equations 4.11, 4.12 and 4.13.

This is similar to our combinatorial problem in Equation (4.9) with multiple power modes (classes) in devices (items) that need to be chosen (binary decision) under constrained aggregator capacity to maximize the utility (profit) of the devices. Additionally, the multiple aggregators (multiple knapsacks) concept can be directly mapped as multiple knapsacks. Hence, it can proved that the problem in Equation (4.9) is NP-hard.

□

4.4 Low-complexity heuristic solution

Due to the nature of the problem, obtaining an optimal solution leads to high complexity since the problem scales exponentially with the number of devices and aggregators. To overcome this issue, we propose an online distributed low-complexity algorithm that can scale well at an aggregator-level.

Aggregators are responsible to handle the scheduling of devices to minimize utility loss. Devices submit their requests to their respective cluster's aggregator in the form of Equation (4.1) and aggregators schedule these requests to minimize the utility loss. Each aggregator computes the priority for every device within its cluster using the remaining time and power to meet the device demand (Equation (4.14)) and sorts them in descending order. Additionally, the power mode for each device is also decided for the time slot and the schedule gets disseminated to all devices within the same cluster indicating the order in which the devices receive power. Using this information, each device computes its corresponding utility loss.

Furthermore, mobile devices that cannot fulfill their demand within the deadline submit their requests to move to another aggregator. Since the aggregators are connected using a backhaul network, each aggregator can compute the cost and the associated time of movement to other aggregators. The backhaul network contains electrical connections to the grid as well as network connections with the grid operator to facilitate load data analysis. Hence, using the cost and computed utility loss for movement provided by the aggregator, mobile devices are able to take independent decisions on making a move to a different aggregator. It is important to note that aggregators do not have any information on the future arrivals of devices and computes a schedule only based on the information in the current time slot.

Priority function. At every time slot, devices get assigned a priority order from the aggregator using a priority function that uses the remaining time to deadline and remaining power to fulfill the demand. Each aggregator computes the utility loss for each device at a given time slot and orders them in decreasing order of loss, i.e., the device with the highest loss gets the highest priority. The priority function pr_k is defined for all devices with $\mathcal{P}_k(t_c) \leq E_T$ as:

$$pr_k = \begin{cases} \dfrac{(E_k - \mathcal{P}_k(t_c))}{E_k} \cdot (t_c - T_k), & (t_c > T_k) \\ \dfrac{E_k - \mathcal{P}_k(t_c)}{E_k} \cdot \dfrac{1}{(T_k - t_c)}, & (t_c < T_k) \\ \dfrac{E_k - \mathcal{P}_k(t_c)}{E_k}, & (t_c = T_k) \end{cases} \tag{4.14}$$

where t_c is the current time slot when the priority is computed. With t_c significantly lower than T_k, devices can sustain waiting without incurring utility loss. Alternatively, when t_c is greater than T_k, d_k has to be served power at the earliest to prevent utility loss. With t_c at T_k, the priority function is the ratio of the remaining demand to complete over the total demand of the device. The priority function is a function of the remaining power and the remaining time to the deadline. Intuitively, this translates to

the penalty from β_m and β_d which yield higher losses due to an increase of demand due to movement and deadline misses, respectively. In case two devices have the same pr_k, the tie is broken using other parameters such as κ_k (higher κ_k gets priority) and feasible α_k (higher feasible α_k gets priority) based on the available aggregator supply.

We also implement a couple of other scheduling algorithms as a baseline: highest power-based scheduling and earliest finish time-based scheduling. The highest power scheduling allows the device with the highest power requirement to be served first. Hence, the priority is a direct function of $\alpha_{k,n}$ at every time slot and is given as:

$$pr_k = (E_k - \mathcal{P}_k(t_c)) \cdot (\alpha_{k,n}), \qquad (E_k - \mathcal{P}_k(t_c)) > 0 \tag{4.15}$$

In contrast to the highest power, the earliest finish time assumes the devices are provided power to maximize the utilization of the aggregator power budget. Hence, it is computed as the ratio of sum of all the remaining energy demand of the devices over the aggregator's power budget, given as:

$$pr_k = \frac{\sum_{d_k \in d_k^j} (E_k - \mathcal{P}_k(t_c))}{\hat{\alpha}_j}, , \qquad (E_k - \mathcal{P}_k(t_c)) > 0 \tag{4.16}$$

The distributed low-complexity algorithm that is executed at every time slot is shown in Algorithm 5. At a network-level (lines 1-6), the aggregators (re-)initialize the devices in their cluster based on device arrival requests and the associated priority list. They also initialize the *status_list* variable to an empty set that is later populated with aggregator utilizations (load). Each aggregator computes the priority for each of the devices within its cluster based on Equation (4.14) and sorts them in descending order. After computing the remaining time and power to meet the device demand, the parameters are passed to the priority function shown in Equation (4.14). Devices in the list are assigned power with the lowest of their feasible power modes since few modes may not be feasible based on the progress, *i.e.,* starting from the highest priority with the least feasible power mode. Subsequently, any available power at the aggregator is used to upgrade the devices to higher power modes in the same order of priority. Lastly, the status_list is updated by communicating the utilization among the aggregators and shared with the devices for mobility. Aggregator steps are illustrated from lines 7-17 of Algorithm 5. At the device-level (lines 18-24), mobile devices that did not receive power use the information from the status_list to compute the loss due to movement. If the loss can be minimized at a neighboring aggregator, it sends a request to join the neighboring cluster. Any received requests are served in the same order - if the existing requests consume the available supply, no new requests are accepted. The above steps are common to all devices independent of their mobility capabilities shown in steps 1-11 in Algorithm 5.

The time complexity of the heuristic is $\mathcal{O}(n \log n)$. $\log n$ is the complexity of the sorting algorithm used for prioritizing devices while repeating it for n devices.

Algorithm 5 Distributed Algorithm to minimize utility loss of devices (static and mobile) with multiple power modes

▷ //Network-Level:
for $a_j \in \mathcal{A}$ **do**
    Initialize status_list $\leftarrow$ []
    Rcv(θ_k) $\forall d_k in a_j$
    Initialize d_k^j
    Initialize $pr\_list_j \leftarrow$ []
end for

▷ //Aggregator-Level:
for $d_k \in \mathcal{D}$ **do**
    Compute $\mathcal{P}_k(t_c)$
    Compute $pr_k \leftarrow$ priority$(\mathcal{P}_k(t_c), E_k, t_c, T_k)$
    Sort_descending $(pr\_list_j(pr_k))$
end for
$p_k(t_c) \leftarrow \alpha_{k,i} \forall d_k \in pr\_list$
if $(\hat{\alpha_j}$ - $\sum_{d_k \in d_k^j} p_k(t_c)) > 0$ **then**
    $p_k(t_c) \leftarrow \alpha_{k,i+1}$ $\forall d_k in pr\_list$
end if
status_list $\leftarrow$ get_agg_status$(\mathcal{A} \setminus a_j)$
Send (status_list) $\forall d_k^j$

▷ //Device-Level:
$c_{a_j \rightarrow a_{\hat{j}}}, \hat{\alpha j} \leftarrow$ Rcv(status_list)
if $(\gamma_{pk} == 0)\&\&(m_k == 1)$ **then**
    for $c \in \mathcal{C}$ **do**
        Compute $\beta_k^m(t_c) \leftarrow$ calc_move($c_{a_j \rightarrow a_{\hat{j}}}$,$\hat{\alpha j}$)
        Compute $\beta_k^d(t_c)$
    end for
    if $(\beta_k^d(t_c) > \beta_k^m(t_c))$ **then**
        Send(θ_k)
    end if
end if

4.5 Experiments

4.5.1 Optimization platforms

We test a real-world problem of load balancing among electric vehicles involving scheduling and resource allocation to validate the efficacy of our proposed clustered architecture. To ensure the practicality of the experiments and the solutions to the above problems, we test the optimization problem in a python-based simulator as well as a standard solver.

Gurobi simulator. The formulated model is implemented using the Gurobi simulator [106]. Gurobi is a standard solver that is well accepted throughout the academic and industry communities. It is available on various implementation platforms including C, C++, C#, Java, MATLAB, Python, VB and R. We adopted Python to conduct our experiments and used multi-objective optimization to solve mixed-integer, non-linear programming.

Python simulator. Although the solver can produce an optimal/near-optimal solution, the runtime of the solver is prohibitive. To mitigate this issue, a heuristic is proposed to solve the optimization and is implemented on a Python simulation [107]. Python 3.7 was used for both Gurobi and the proposed heuristic. An object-oriented approach using classes was adopted to allow future expansion for easier integration of additional elements.

Background on synthetic data generation. The UUnifast algorithm [108] has been widely used for processor task set generation with uniform distribution for computation in real-time systems. The purpose of the UUnifast is to generate unbiased exhaustive tasksets across the range of input parameters. UUnifast in its simplest form generates n different utility values given a total utility U in the range of $0, U$. In our example of smart grids, Uunifast can be used for a single source (aggregator) with n devices. However, for our model that uses distributed energy resources (DERs), we would also need to specify constraints on each energy source. Hence, the synthetic data for device demand taskset in our experiments was generated using the Dirichlet rescale (DRS) algorithm [109]. DRS can get generate a variety of tasksets with different utilization proportions. For example, if there are 12 tasks to be generated with 25% high utilization [1-1.2], 25% medium utilization [0.75,1] and 50% low utilization [0.25-0.5] with total utilization of 6.5 or more (sum of lower bounds), DRS is able to generate 3, 3 and 6 tasks for the corresponding utilization bounds, respectively. The generated tasks conform to the specified bounds and are uniformly distributed across the range of bounds (unbiased).

Experimental setup. In this section, we evaluate our heuristic solution in minimizing the utility loss with a synthetic and a real-world dataset. As discussed in Chapter 2, we compare the performance of the proposed algorithm with a solution of the optimization problem in the Gurobi solver as a baseline [106]. The synthetic data was generated using the DRS algorithm [109] based on the parameters defined in Table 4.2. The parameters

Table 4.2: Parameters used in our experimental setup for evaluating the proposed algorithm.

Parameter	Value
$\mathcal{A}$	5
$\hat{\alpha}_l$	$500k$Wh
No. of Devices	{20, 40, 60, 80, 100}
No. of Timeslots (τ)	50
Deadlines/Periodicity	{6,12,24,48}
Classes of aggregator load	{ [0.5-1] (L), [1-1.25] (M), [1.25-1.5] (H) }
Aggregator load combinations	[L,L,L,M,H],[L,L,M,M,H],[L,M,M,M,H],[L,L,M,H,H]
T_0	0.5h
$(\delta_{a_j \to a_{\hat{j}}}, t_{a_j \to a_{hatj}})$	{(0,0), (1,0.15), (2,0.15), (3,0.15), (4,0.15)} (slots, kWh)
α^{min}	1kW
$\{\|\alpha_k\}$	{1,2,3,5,10,20,50}kW
Proportion of Mobile Devices	$\{25\%, 50\%, 75\%, 100\%\}$
κ_k	{1.6, 1.8, 2.0}

used in Table 4.2 are derived from the real-world datasets to mimic devices such as HVACs [110] and EVs [111].

Each cluster load was split into three classes: Lightly loaded, medium loaded and heavily loaded with utilizations shown in Table 4.2. This indicates that the devices use up the corresponding utilization of the aggregator capacity, *i.e.,* for 0.8 utilization and 500kWh as aggregator capacity, devices consume 80% (400kWh) of the aggregator capacity throughout the time horizon ($\mathcal{T}$). Increasing the utilization to the higher end (H) on all aggregators leads to a non-feasible schedule since each device would have high demand, leading to some devices (low-criticality) never getting scheduled. Devices are generated to achieve utilization by taking into consideration the maximum consumption capacity of various device classes (HVAC, EV, etc.). For example, devices such as iron consume 1kWh, airconditioners consume about 3kWh and EVs can consume up to 50kWh. A deadline is randomly chosen to create a periodic demand (new arrivals after a period, *i.e.,* arrival = deadline) from the devices and corresponding power modes are assigned to ensure that demand can be met. Mobility costs and movement time between the clusters are inversely related to distance *i.e.,* the farthest cluster incurs the highest mobility cost and time for movement (4, 0.15) and vice-versa (1, 0.15). Distances are proportional to the movement time between the clusters.

4.5.2 Heuristic performance

To test the performance of the heuristic in an unbiased way, an exhaustive synthetic dataset was generated using the DRS algorithm [109]. Based on parameters from Table 4.2, 50 samples of each aggregator class combination with different device utilizations are generated and simulated. The simulation results are shown in Figure 4.5. Axes of

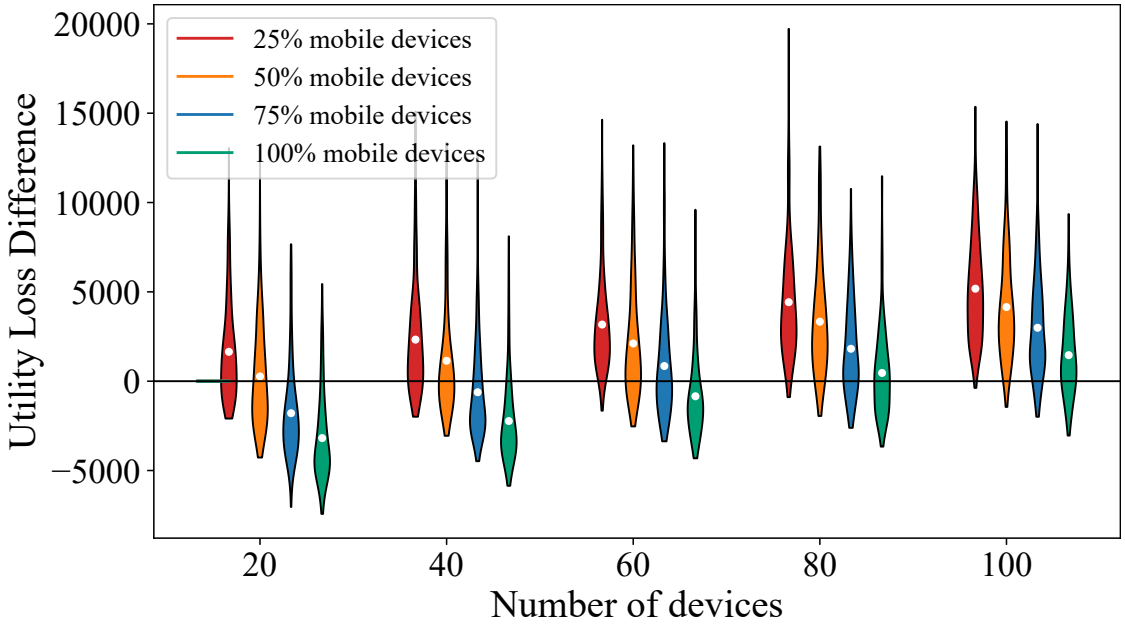


Figure 4.5: The difference of utility loss achieved with and without mobility

the violin plot represent the utility loss plotted as a function of the number of devices utilizing the aggregator capacity as listed in Table 4.2. Each violin in the plot represents the difference of utility loss difference between device mobility enabled and disabled, *i.e.*, $u_k(t)(\text{mobility}) - u_k(t)(\text{non} - \text{mobility})$. The former allows devices to move across different clusters while the latter restricts all devices to their original cluster. The length of the violin indicates the variance while the average value is indicated by the white dot at the center; the sharp peaks represent the outliers.

The observed loss with mobility-enabled was higher than that of mobility-disabled for a lower number of devices and a reversed pattern was observed for a higher number of devices. For a fixed capacity of the aggregator, a lower number of devices translates to devices with higher demand (*e.g.* electric buses, etc.) while a higher number of devices have devices with lower demand (*e.g.* HVACs, e-scooters, etc.). In the case of fewer devices, movement may not be effective to reduce the utility loss as the higher demand from the devices may not be available at any aggregator. Scheduling a mobile device with high demand (and associated power modes) is difficult at the aggregator as devices within its cluster also have a high demand resulting in long wait times. Consequently, there is a higher utility loss due to mobility and deadline misses at the devices based on the resulting schedule.

As seen from Figure 4.5, with limited devices, the demand reduces from each of the devices, giving a higher probability of allocating the residual power to a new device. This can be seen by the reduction in utility loss for mobility enabled in the plot for a higher number of devices. The scheduling can be realized practically as electric buses are more difficult to schedule than a low-power EV or electric scooter.

4.5.3 Comparison with solver

Typically, optimization problems are solved with a solver since a "near-optimal" solution can be obtained. Although a solver can produce an optimal solution in practice, the runtime to complete the simulation is prohibitive. In this experiment, we compared the performance of the proposed heuristic with the solutions obtained from the Gurobi solver. In our tests, even with a runtime upwards of 10 hours, the solver could not find a solution for some parameters due to large number of parameters.

As the solver takes a prohibitive time to complete the simulation, a time limit is set to the maximum time taken by the heuristic solution to terminate the simulation to obtain a feasible solution. Additionally, we randomly picked 10 out of 50 samples across all classes in the previous experiment to test the performance of the solver. Configuring the solver with the heuristic timing allows for a practically feasible time that aids in managing wait times (to obtain a schedule) for the devices. We compare solver and heuristic solutions with the difference in the utility loss of solver and heuristic with mobility enabled, *i.e.,* $u_k(t)(\text{solver}) - u_k(t)(\text{heuristic})$. With each time slot mapping to 30 minutes, the maximum observed time from the heuristic allows for a schedule computation from the solver without incurring a long wait time for devices.

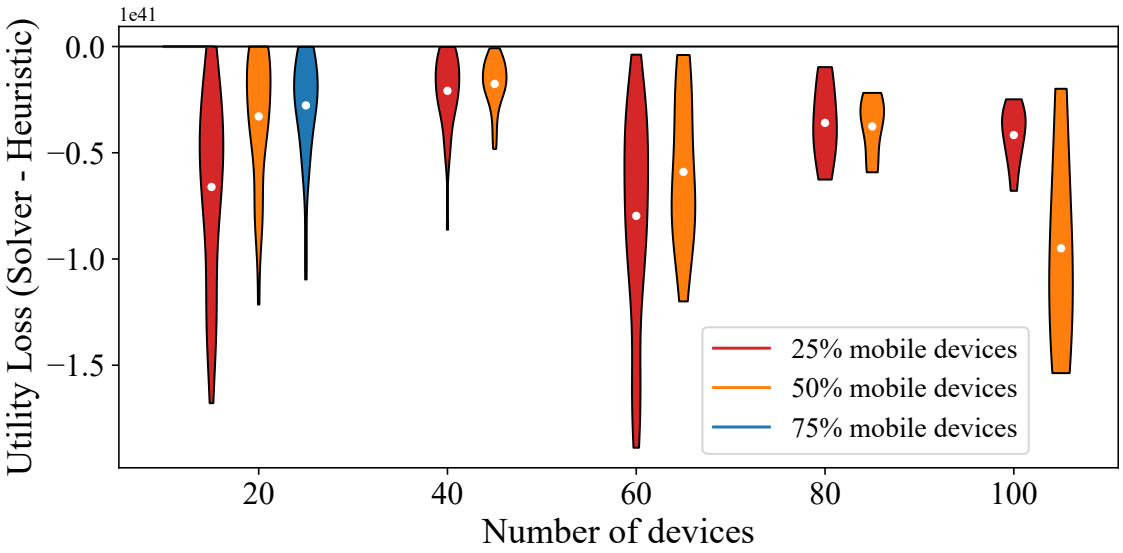


Figure 4.6: Comparison of solver performance with the heuristic for the same runtime.

From Figure 4.6, it can also be seen that the increase in number of devices also increases the losses from the solver. On an average, the utility loss from the solver is worse off by a factor 10^{39} than the heuristic solution. Since the solver needs to compute several stages such as pre-solve, barrier, etc. before the start of optimization, the limited runtime restricts the solver depending on the number of variables. Table 4.3 lists the number of samples that could produce at least one feasible schedule for the different number of devices along with the proportion of mobile devices among them. With a lower number of devices and a lower proportion of mobile devices among them, the model is able to

Table 4.3: Number of samples generated for the solver.

No. of devices	No. of Samples (% mobile devices)		
	25%	50%	75%
20	40	40	5
40	40	13	0
60	40	6	0
80	40	7	0
100	40	9	0

Table 4.4: Utility loss for EV dataset with different strategies.

Proposed Heuristic	Earliest Deadline	Highest Power
1606.63	3939.27	3757.18

produce at least one solution due to lower complexity. As the proportion of mobile devices increase, the number of solver variables and the resulting losses are exacerbated. This is seen from Table 4.3 when 75% of all devices were mobile, only 20 devices had a few samples that could complete. No sample for 100% mobile devices could complete the run as the time limit expired before finding a solution.

4.5.4 Evaluation on a real-world dataset

We also simulated the heuristic and the solver on a real-world dataset derived from the EV testbed at Caltech [30]. The complete charging dataset of the year 2020 was extracted from the testbed data along with data from Table 3.2 for missing parameters such as power modes, etc. The arrival times were mapped to a single day (48 time slots) as there was no congestion found with a limited number of vehicles.

The results are shown in Table 4.4. We also tested the dataset with two standard scheduling algorithms: earliest deadline schedules devices with the nearest deadline to the current time slot while highest power schedules devices with the highest demand. The solution of the proposed heuristic is 59.21% and 57.23% better than the earliest deadline and highest power scheduling algorithms, respectively. The solver could not produce a solution even after 8 hours while the slot time duration (τ) is 30 minutes. This duration (8+ hours) is impractical for any device arriving with a deadline to know if it can be scheduled, *e.g.,* an EV charging overnight waits without a schedule. The heuristic produced a solution in one minute with a per slot average time of 1.25 seconds showing the practicality of our proposed solution.

4.6 Summary

In this chapter, we show the feasibility of device mobility between clusters managed by aggregators. We exploit the device properties such as power modes and mobility to address the issue of load balancing and scheduling. Existing demand response solutions

in smart grids do not integrate different power modes of the devices as well as their mobility property. Our model integrates these properties to estimate the utility loss incurred due to delays in receiving the power. We proposed an online low-complexity heuristic to minimize the utility loss with a practically feasible runtime compared to a solver with similar runtime. A real-world testbed data was also used to verify the model and obtain a feasible schedule. Our solution also performed better by over 57.23% on a real-world dataset compared to existing scheduling solutions.

5 C-sync: Energy-efficient and resilient time synchronization using clustered networks

Chapter 3 introduced an energy-efficient clustered architecture for IoT nodes as communication among these nodes forms the bulk of their operation to exchange information [5]. However, the use of RDC and CSMA-CA require an exchange of multiple messages till a message is successfully delivered across the neighboring nodes. To minimize the number of messages exchanged, a common notion of time is essential among the nodes. Time synchronization is an efficient and effective method to minimize communication significantly, allowing nodes to operate in a low power mode except during transmission. Synchronizing all the nodes' local hardware clocks to a common global time facilitates communication at specific instances and, thereby, provides an infrastructure for scheduling. Applications such as real-time systems start with a time-synchronized network as the foundation for communication among the nodes. Additionally, with limited communication during time synchronization, the information exchange among the nodes must be trustworthy. In this chapter, we present *C-sync*, a novel clustering-based resilient time synchronization protocol.

A time synchronization protocol needs to ensure stability in synchronization throughout the network. Wireless sensor networks are often plagued by error-prone nodes that can result in faults including, but not limited to, selective forwarding and tampered data (spikes, outliers, etc.) [35]. These faults form a sub-class of byzantine faults observed in radio communication and could result in major deadline misses and power dissipation due to erroneous time information, leading to destabilization of the network and significant message losses. A faulty node with incorrect information can jeopardize the entire network with false information if not addressed [112]. Most importantly, synchronization protocols must be resilient to such faults and ensure functional correctness to minimize potential network downtimes.

A faulty node with incorrect information could influence the network in existing synchronization solutions, albeit providing the basic deterrence against faults since there is no verification of the data. For example, a critical sensing application with distributed logging of events may be rendered useless if an event's timestamp is recorded differently by a few devices in the neighborhood of the event. Erroneous information disrupts the analysis and debugging of data. Hence, there is a need to integrate fault tolerance while designing a synchronization protocol to ensure that the error from the time source/connecting node does not impact the entire network.

Some of the existing protocols handle node failures by switching to a different node to provide a reference clock [21, 14, 22]. Although reference node switching could work

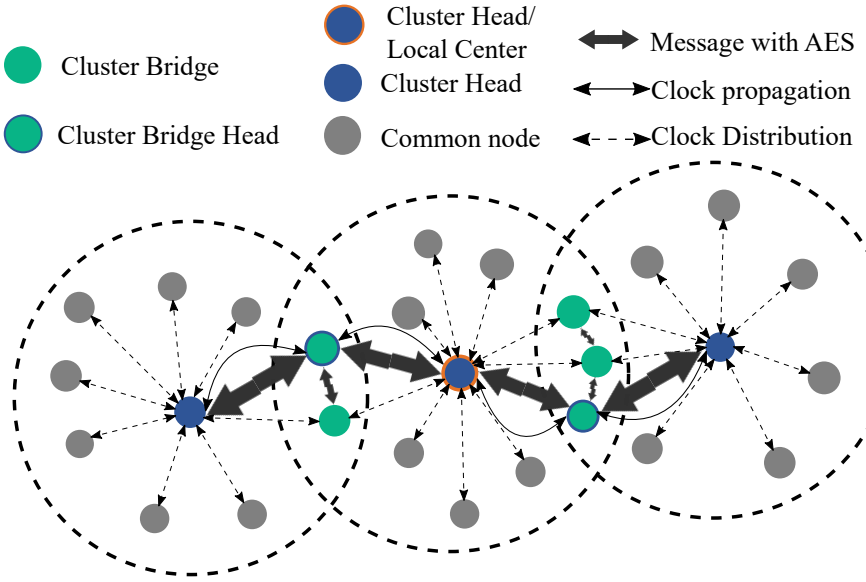


Figure 5.1: Time Synchronization in C-sync using a clustered architecture for power efficiency and fault resilience.

as a tentative solution, an adequate fault handling mechanism is absent. For example, faults such as selective forwarding in a reference node could send all but critical messages resulting in desynchronization without any means to fix the issue. Generally, protocols reliant on a reference node require additional measures to mitigate the impact of a compromised reference node. In the absence of reference nodes, some protocols adopt a decentralized design that has inherent fault resilience [15, 113, 114, 26]. Faulty nodes are excluded from information transmission during synchronization if their information is substantially different from those of other neighboring nodes. However, managing the faulty nodes in a decentralized network exponentially increases the messages exchanged and consequently, the power consumption.

From Chapter 1, we have seen the requirement of three primary goals: energy efficiency, adaptability and fault resilience. While all three properties are needed in time synchronization solutions, the current solutions mostly focus only on the energy efficiency and adaptability aspects with an additional property of synchronization accuracy. We aim to fulfill the standard requirements of synchronization protocols as well as ensure resilience to faults in the network. To achieve energy efficiency, these solutions minimize communication (radio "ON" time) such that the least number of messages are exchanged to achieve and maintain synchronization. Synchronization accuracy varies from a few seconds [115] to a few microseconds [22, 14, 15], depending on the type of protocol and application used. For sensors in real-time systems, the expected accuracy is in the order of microseconds [5]. Typically, achievable accuracy is bounded by the resolution and the stability of the clock used by the node. Adaptability ensures that the functionality and the performance of the protocol are not impacted by the growth in network size when new nodes are added. Although existing solutions minimize the im-

pact of adaptability through flooding, decentralized averaging, etc., the synchronization error remains directly proportional to the number of hops between the time source and the synchronizing node. It is desirable to minimize the distance of nodes to the source of time information such that the error between the source and the synchronizing node does not scale with the network size.

Another important attribute required for a synchronization protocol is the ability to ensure reliable time information is transmitted by the nodes so that a stable synchronization is maintained throughout the network. A node that shares an incorrect/faulty clock value at any given time slot can jeopardize the entire network causing significant message losses and power dissipation. Some of the existing protocols make provision for node failures by switching to a different node to provide a reference clock [22] or excluding them during synchronization [15]. However, this leads to a delay in re-synchronization and a temporary reduction in synchronization accuracy. Hence, it is vital to factor in the resilience to faults while designing the protocol to ensure that the error between the source of the clock and the receiving node does not impact the entire network.

In this chapter, we propose a decentralized clustering-based time synchronization protocol as shown in Figure 5.1 (referred to as *C-sync*) to ensure fault resilience of the network in addition to the standard three metrics of synchronization protocols. The nodes, also referred to as Common nodes (CM) elect a representative node called the Cluster Head (CH), to which the nodes associate to form clusters based on properties such as the degree of a node (number of communicating neighbors). The CH node is connected to other CH nodes of neighboring clusters through a few Cluster Bridge (CB) nodes that ensure information propagation between clusters. Further, to prevent all the CB nodes from communicating (*i.e.,* to avoid channel interference), a representative CB node called the Cluster Bridge Head (CBH) is elected to manage the communication. The other CB nodes are used in the event of a fault in the network. We address CB and CBH interchangeably for all scenarios except for fault handling. C-sync leverages the clustered architecture to contain the faults (fail-stop and a subset of byzantine faults) from spreading beyond a cluster and to let nodes remain in sleep mode for a longer time, thereby reducing the power consumption due to the radio significantly. As CH and CB nodes wield greater influence on the information propagated within and across clusters, it is critical to identify faulty nodes quickly to prevent erroneous information from spreading across the network. We design a consensus mechanism among a group of nodes in the neighborhood to verify the sender and its information to ensure correctness and integrity in the transmitted data. More information on the fault model and the fault handling in C-sync is explained in Section 5.3.

In C-sync, a concept called Local Centers (LC) is introduced to effectively handle device dynamism, where some CH nodes within the network are elected as reference nodes. LCs introduce a parametric (deterministic) restraint on the hops any node has to go through to obtain time information from the source node. These nodes coordinate the distribution of time information to maintain synchronization throughout the network such that the synchronization error is limited by restricting the number of hops between LCs and other nodes. The resilient design of C-sync coupled with low power consumption and adaptability enables the design of real-time applications in decentralized systems.

Table 5.1: Comparison of prominent synchronization solutions in the literature.

Characteristics/ Solutions	FTSP [21]	PulseSync [14]	Consensus [116]	Glossy [22]	GTSP [15]	C-sync
Central reference node	Yes	Yes	No	Yes	No	No
Messages transferred	High	High	High	High	High	Low
Complexity	High	High	Low	High	Low	High
Robustness to network changes	Low	Low	High	Low	High	High
Accuracy	Low	High	Low	High	High	High
Time to Synchronize	High	Low	High	Low	Low	High

LCs are further explained in detail in Section 5.2.2. To summarize, the contributions of this chapter are as follows:

1. We propose C-sync, a decentralized fault-resilient clustering-based time synchronization protocol suited for large-scale IoT networks. C-sync introduces multiple time sources in the network to constrain the error between any node and its reference.

2. We show that the proposed protocol achieves synchronization with significantly lower power consumption while ensuring accuracy is not compromised.

3. Through extensive experiments on a real testbed and theoretical analysis, we show the fault recovery mechanism of C-sync and show the performance with power efficiency and synchronization accuracy.

Organization. A comparison of the current solutions for fault-tolerant time synchronization is presented in Section 5.1. A detailed description of the C-sync protocol is provided in Section 5.2. Section 5.3 outlines the considered fault model and the fault handling mechanism of C-sync. The experimental setup and the experimental results are discussed in Section 5.4. Section 5.5 concludes this chapter.

5.1 Related work

The existing literature can be classified based on the availability of fault handling mechanisms in the protocols. Protocols with dedicated fault-tolerant features are described under secure synchronization solutions while the rest of the protocols are grouped together as generic synchronization solutions. We discuss different works of literature classified into these two categories:

Secure synchronization solutions

Ganeriwal *et. al.* investigated the problem of secure time synchronization using shared key encryption and delay threshold-based detection [117], *i.e.,* the delay between the estimated and the actual time. However, synchronization is done only when the delay between the estimated and the actual message falls below a threshold and assumes that key sharing is in a secure environment. Li and Rus counter byzantine faults by adding a layer of cryptographic encoding and decoding during message exchanges but suffer from long convergence times [118]. Blockchain-based protocols [27, 119] achieve resilience against byzantine faults at the cost of low accuracy (in the order of seconds) and high computation overhead. Max-consensus was used to estimate the clock difference to a threshold value to detect byzantine nodes but achieves a low synchronization accuracy [26]. Sundial [16] was proposed for fault-resilient synchronization between data centers by combining a hardware-based detection and software-based reconfiguration for hardware to handle faults. However, Sundial requires resource-rich hardware for fault detection while taking substantial time for reconfiguration change transmission from the reference node to an actual change in hardware. Temporal correlation of messages was used between neighboring nodes to correct synchronization errors [18]. However, error detection and correction require the exchange of a significant number of messages resulting in communication overhead. While assuming a trusted resource-rich reference node, digital signatures and message filters were used as validation tools [12]. Secure synchronization protocols focus on ensuring resilience to faults but do not cater to accuracy and energy efficiency due to complex fault handling mechanisms. Additionally, most of the works on secure synchronization solutions are centralized with the assumption that the reference node cannot be faulty. Furthermore, they have not been hardware-proven.

Generic synchronization solutions

Historically, the Global Positioning System (GPS) or Network-Time Protocol (NTP) [115] has been used for time synchronization in networks. However, these protocols are not applicable for resource-constrained nodes. Although recent protocols make use of these techniques for synchronization [120], their applicability is highly restricted in resource-constrained environments. Although recent protocols and hardware with long-range communication are able to utilize these resources for synchronization [120], access to GPS or internet time is highly limited in resource-constrained environments. Reference Broadcast Synchronization (RBS) [121] and Time-sync Protocol for Sensor Networks (TPSN) [5] were one of the first works achieving receiver-side and peer-to-peer synchronization. Reference Broadcast Synchronization (RBS) [121] synchronized a set of receivers to minimize sender-side uncertainties while Time-sync Protocol for Sensor Networks (TPSN) used peer-to-peer synchronization with MAC-layer timestamps for both sender and receiver nodes [5]. However, both protocols cannot handle ad-hoc networks and have no compensation for clock drifts. The absence of clock drift compensation and high communication overhead in TPSN was addressed by Flooding Time Synchronization Protocol (FTSP) [21] using periodic network flooding of synchronization messages originating from the reference node. Flooding is a process of rapid dissemination of

messages through the network. Receiving nodes in FTSP use linear regression on the received messages to compute clock drift to minimize errors. Yildirim and Kantarci [122] leverage FTSP to improve the accuracy of flooding by restricting linear regression to messages received only from single-hop neighbors for offset compensation while using the least-squares method for clock drift compensation. Offset is the difference in clocks between the sending and the receiving node. The time to complete synchronization and adaptation to changing topologies are drawbacks of their contribution. Pulsesync [14] uses fast flooding by immediately sending a synchronization message upon receiving one, to reduce the flooding latency and the synchronization error for the farthest node using a breadth-first-search tree. Meanwhile, Glossy protocol achieves synchronization using constructive interference of modulated signals with a temporal displacement within a threshold [22]. This necessitates nodes to be equipped with high-quality radios with low noise and distance between nodes to achieve the synchronization threshold. All the above flooding synchronization schemes require a large number of messages from a central reference node that distributes the time information to the rest of the network. Additionally, centralized solutions have a single-point failure when the reference node fails, leading to some downtime before a new reference node gets elected. It is important to note that the inherent reliance on the reference node by all nodes of the network leads to a single-point of failure and constant re-configuration in presence of faults. The delay due to regular re-configurations could be catastrophic in critical real-time applications such as electric grids, etc. [112]. Decentralized solutions do not rely on a single reference node to achieve time synchronization. To this end, Gradient Time Synchronization Protocol (GTSP) [15] synchronizes precisely among the neighbors by estimating a global clock formed by an average of drift and offset among 1-hop neighbors. Based on the network topology and placement of nodes, clustering coupled with consensus has been proposed to synchronize the nodes [113, 114]. Wu *et. al.* use the LEACH [23] clustering protocol which assumes a synchronized network for communication while Wang *et. al.* assume a fixed topology with a fixed state for all nodes without any communication delays. Both cluster-based protocols are not resilient against faults and cannot adapt to dynamic changes in the network. Emergency Broadcast Slot (EBS) [123] synchronizes decentralized networks but has a high dependency on a minimum set of neighbors to be active to achieve and maintain synchronization.

Other synchronization solutions include the method of average consensus from control theory [124] was used to individually estimate the clock drift and the offset [116]. The use of two individual consensus mechanisms is compute-intensive and has a high power consumption. [125] uses max-consensus multiple times for reduction in computation. The number of iterations depends on the diameter of the network. [126] seeks to solve average consensus which does not have a linear consensus function. It limits the maximum value transmitted to improve energy efficiency by using the derivative of the non-linear function and achieves convergence. Although the message size is reduced due to derivatives, there is an increase of computational complexity than the standard consensus. However, assumptions such as zero transmission delay and lossless medium are not practical and cannot be translated to hardware. [127] and [116] use consensus to arrive at a global skew and offset. They use average consensus to estimate a global

logical clock that all the network nodes adhere to. Xie et. al. aim to achieve consensus in finite time using the geometric mean of local clock drifts to compute the global clock drift and subsequent offset compensation [128]. Since a tree topology is assumed, it takes a long time to complete synchronization for the last leaf node. Although synchronization in the consensus mechanism completes in a fixed period, there is a continuous correction of the clock leading to high communication overhead. Authors in [129, 125] conclude that consensus can be reached if the underlying directed graph has at least one spanning tree at every synchronization cycle. Although the method is optimized in computation and memory, for all consensus methods above, the time to converge and power consumption are high due to continuous transmission of messages. In addition, the network is also expected to be static with no changes in topology. Authors in [130] aim to reduce the number of messages exchanged in the network during consensus. Since the error in average consensus among the nodes converges to zero asymptotically over time, the messages exchanged are reduced as the error inches towards zero. Each node has a threshold of the difference between current and previous values beyond which nodes stop broadcasting messages. The reduced messages are at a cost of reduced accuracy in synchronization. Although synchronization in the consensus mechanism completes in a fixed period, they are compute-intensive with a growing consensus convergence time as the network scales.

Alternate synchronization solutions

In [131], Cho *et al.* follows a two-level hierarchical structure with a master node sending controlling the synchronization operation. Synchronization takes place with a central node sending messages frequently to have all nodes follow a common clock. The hardware testbed is a Field-Programmable Gate Array (FPGA), a resource-rich hardware. Viswanathan *et al.* [132] rely on the availability of electric grid voltage signals and utilize the time information in signal fluctuations of periodic voltage signals. The availability of grid voltage signals is restricted to industry-grade devices since it is not applicable for battery-powered or DC voltage nodes. The use of grid voltage signals enables the solution to achieve good accuracy as well as scale for long distances. A combination of a high-speed and low-speed clock called VHT (Virtual High-resolution Timer) is used to achieve synchronization of the high-speed clock with the low-speed one [133]. The authors propose to selectively turn off the high-speed clock during sleep and as the low-speed clock is always turned on. The methods incur additional overhead in hardware. An issue of combined skew surfaces when there are multiple clock interferences during communication between nodes. For all the methods described above, there is a common source of the clock to achieve synchronization. A failure of the source would lead to the failure of the overall synchronization. Other solutions add to hardware overhead with specialized circuits and timers to reduce the jitter from the existing hardware clocks [133, 134].

A summary of some of the prominent works in literature classified based on different properties is provided in Table 5.1. A qualitative comparison is shown as it is challenging to establish an evaluation environment for every protocol to provide a quantitative comparison. Among the available synchronization solutions, the GTSP protocol has an

inherent fault tolerance due to averaging of individual time information, *i.e.,* a single faulty node does not impact the overall average. Additionally, GTSP is the only decentralized protocol with high accuracy for generic wireless networks that had been proven on hardware. However, GTSP suffers from high power consumption due to the continuous exchange of messages with every neighbor during synchronization with a wider network impact in presence of faults. Our proposed solution, C-sync, has a slightly longer initial convergence time due to the overhead of establishing the clustered architecture from a completely decentralized structure as discussed in Section 5.2.1. Exploiting this network structure, C-sync can achieve significant energy savings by a limited exchange of messages for maintaining synchronization and handling faults. A fault in C-sync gets isolated to the specific cluster/clusters within which nodes can operate without a reference or get a new reference node without impacting the rest of the network. Hence, we chose to compare C-sync against GTSP as the state-of-the-art decentralized protocol for the synchronization accuracy and energy efficiency on a hardware platform as presented in Section 5.4.3. To the best of our knowledge, no time synchronization solution achieves similar fault resilience with energy efficiency as C-sync. C-sync also exploits clustered architecture to limit the synchronization error of any node in the network to the reference time source.

5.2 C-sync protocol

In this section, we introduce our protocol C-sync and discuss the synchronization mechanism. We provide a pseudo-code to demonstrate its operation and show the election process of local centers with an example network shown in Algorithms 6 and 7. C-sync also provides a unique feature that limits the distance of any node from the time source that is further explained in Section 5.2.2.

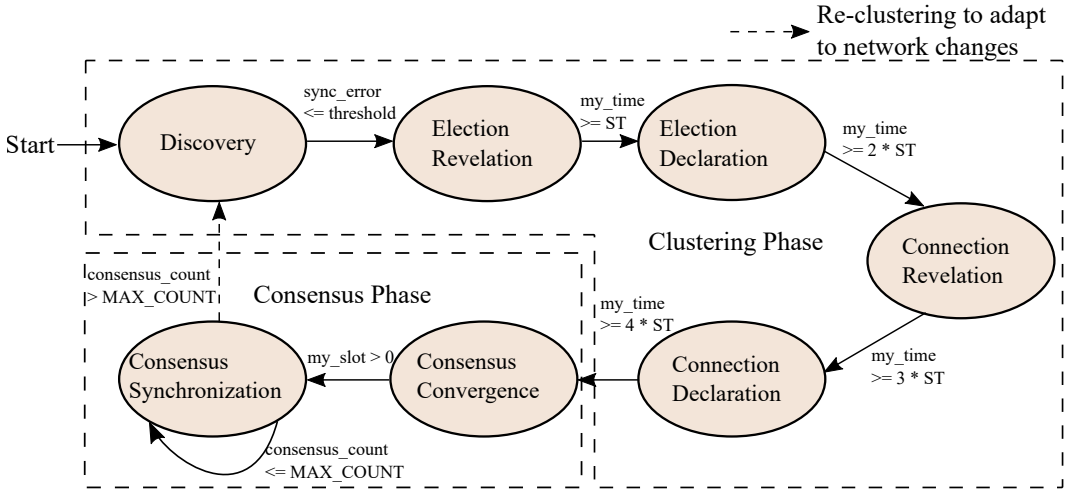


Figure 5.2: The two phases of the C-sync protocol represented as a state machine.

The C-sync protocol follows a 2-phase process as described in the state machine in Figure 5.2. Clustering is the first phase of C-sync derived from the existing clustering scheme DeCoRIC discussed in Chapter 3. DeCoRIC uses a clustering scheme based on the degree of a node, *i.e.,* the node with the highest degree forms the representative CH node that facilitates routing information within and across clusters. The degree of a node is the number of active communication links of a node with its neighbors. DeCoRIC is adapted to the synchronization process to establish the underlying architecture in the network. The state machine from DeCoRIC is transformed with additional states to integrate scheduling between the states upon achieving a loose synchronization after the Discovery state by utilizing the time information in the messages. Once the clusters are formed, the consensus phase maintains synchronization among the clusters by adopting specific time slots for all the CHs and CBs to exchange time information periodically among each other.

5.2.1 Clustering phase

The clustering phase comprises five states to establish the clustered architecture among the nodes. The clustering phase is a process of establishing a loosely synchronized clustered architecture using DeCoRIC (discussed in Chapter 3) while adding time information to the messages transmitted. Since we assume an ad-hoc network with the same wireless channel, any node that joins the network in the midst of the state machine is able to join the nearest cluster and jump to the associate time from the CH. The new node may become a CH/CB node depending on its position in the subsequent clustering phase.

Discovery state. In the discovery state, each node broadcasts messages with the time information derived from its hardware clock and listens to other nodes in the neighborhood to discover its environment. In this state, the logical (*i.e.,* global) clock is the same as the hardware clock and the degree of every node is set to 0. The logical clock $L(t)$ and its parameters (rate and offset) is derived from the hardware clock value $h(t)$ compensated by the rate parameter and an offset value as in [15]:

$$L(t) = \int_{\tau=0}^{t} h(\tau) l(\tau)\, d\tau + \theta(t) \tag{5.1}$$

where $l(\tau)$ is the logical clock rate relative to hardware clock (*logical clock rate* in short) and $\theta(t)$ is the logical clock offset. The logical clock rate is the average of relative rates to all its neighbors, whereas the logical offset is the average of relative offsets to all its neighbors.

For a node i with n_i neighbors, the clock rate l_i and offset θ_i is computed using

$$l_i(t+1) = \frac{\sum_{j \in n_i} l_j(t) + l_i(t)}{n_i + 1} \qquad \text{and} \tag{5.2}$$

$$\theta_i(t+1) = \theta_i(t) + \frac{\sum_{j \in n_i} L_j(t) - L_i(t)}{n_i + 1}. \tag{5.3}$$

Clock rate represents the rate of change of clock offset over time. However, it is important to note that the absolute logical clock rate for one's own logical clock is the ratio of the relative logical clock rate and the hardware clock.

Upon reception of a message from a new neighbor, the receiving node increments its degree and stores the time information (rate and offset) from both logical and hardware clocks of the sender. The information of the neighbor includes its degree, clock offset and clock rate of the logical clock relative to its hardware clock. It also stores the sender's logical and hardware clocks for computing the logical clock value at a later time. All the messages in this state are asynchronous since there is no reference clock. The MAC layer uses CSMA-CA with message timestamps *i.e.,* MAC-Layer timestamps to minimize the collision and interference in the network as the nodes are asynchronous at the start. The logical clock rate and offset of a node are derived from averaged relative clock rates and offsets of its neighbors. Since all nodes perform averaging for the offset and rate, the nodes are loosely synchronized. The average network delay gets factored into the offset as the logical clock offset gets calculated independently at each node based on the neighbor offsets. If a node can achieve both the offset and rate of its logical clock within a pre-defined threshold, that node creates a reference time (delay) for a state transition to the election revelation state for all nodes in the neighborhood. The threshold is set based on worst-case CSMA backoff to ensure that nodes are able to receive messages even if they are loosely synchronized. The threshold also allows nodes with large offsets to jump to the transition interval based on a synchronized neighbor, preventing long convergence times. Lines 1-18 of Algorithm 6 show the operation of the discovery state and the transition to the election revelation state. Hence, the discovery state allows the nodes to get information on their neighboring nodes.

In the event of a late-discovered node due to the dynamic nature (new/mobile nodes) of the network, the new node listens to messages from the neighboring nodes and associates itself with the nearest CH node as a Common node (CM) of its cluster. Upon receiving the state transition announcement, the CM node directly jumps to the current state of the network. However, it is important to note that the subsequent discovery state may elevate the status of the CM node to a CB or a CH depending on its position in the network to form more optimal clusters.

Election revelation state. The election revelation state facilitates the exchange of degree information from all nodes for the election of Cluster Heads (CHs). The configuration could be changed to use residual energy of the nodes to ensure uniform distribution of energy among the cluster nodes. CHs are representative nodes with the highest degree in their neighborhood and facilitate routing of messages from within its cluster to other parts of the network. CH nodes retain the CM nodes in sleep mode except during transmission, thereby significantly reducing energy. All the nodes are active during this state and update their neighbors' degrees based on received messages. Beyond the creation of reference time from the previous state, all nodes transition to further states at a pre-defined interval of time called the *state transition* (ST) interval. The ST interval is a combination of the error threshold used in the discovery phase and frame

length to ensure reduction in the wireless channel interference and a successful message transmission. All nodes transition to the election declaration state after the ST interval.

Note that the slotted ST intervals are based only on the synchronized time without any dependency on the channel parameters. Additionally, the ST intervals are generated based on the hardware timers of a node without any dependency on software time slots. Although integrating channel information would allow the use of C-sync in Time Slotted Channel Hopping (TSCH) networks, we focus on time synchronization on the same channel for all nodes. The use of the same channel enables a plug-and-play interface with greater network flexibility in contrast to configuring every node.

Election declaration state. Nodes with the highest degree declare their CH status and form clusters. If a node receives a message with a higher degree from its neighbor, it cancels its broadcast and associates itself to the CH node as a common node (CM). CH nodes are consolidated in this state and they form clusters together with their associated CM nodes. All nodes have their radios on in this state and transition to the next state after the ST interval. Operations of election revelation and election declaration states are shown in lines 19-31 of Algorithm 6.

Connection revelation state. Connectivity among the clusters is established through the election of *Cluster Bridge* (CB) nodes that connect two or more CH nodes. In this state, nodes that have more than one CH in their neighbor list transmit while all other nodes remain in sleep mode. The degree information (number of CH neighbors) of CB nodes is extracted by other CB nodes from the received messages. CB nodes play an important role to ensure fault-free dissemination of messages during the consensus phase. Hence, to authenticate and prevent nodes to falsely declare themselves as CB, the messages in this state use AES encryption with the IDs of the CH nodes as the key as shown in Figure 5.1. This encryption is in addition to the already encrypted messages of IEEE 802.15.4 standard to ensure the authenticity of CB nodes [3]. This prevents any non-CB nodes from falsely electing themselves as CBs. The elected CB nodes aid fault detection of all node types even if a few faulty nodes are elected as CH. The ST interval transitions the network to the connection declaration state upon its expiry.

Connection declaration state. Similar to the election declaration, CB nodes (CM nodes with multiple CH connections) declare their status to the CH nodes in the connection declaration state. As with the CH election, CBs with the highest degree or highest address (in case of the same degree) declare themselves as the representative Cluster Bridge Head (CBH) and disclose their neighboring CH nodes. The other CB nodes ensure that the declaration is made by the legitimate CB node by checking the IDs of the CH nodes in the AES encrypted message. The CH nodes discover their neighboring CH nodes through their CB(s). This information is used in the consensus phases to identify the slots during which a CH/cluster synchronizes. All nodes have their radio on from discovery till the connection revelation states, while only the CH and CB nodes are active in the connection revelation and declaration states. The steps involved in connec-

Algorithm 6 Pseudo-code representing the operation of Clustering phase in C-sync.

```
1: INITIALIZE Neighbors ← ∅
2: my_state ← DISCOVERY
3: Msg.Id ← my_addr; Msg.time ← my_time
4: my_CH_count ← 0; consensus_count ← 0
5: broadcast(Msg)
6: if rcv() then
7:     for n in Neighbor.list() do
8:         n.time ← rcv().time
9:         if n.synced() then
10:            my_time ← n.time
11:            my_state ← ELECTION_REVELATION
12:        end if
13:    end for
14:    if n ∉ Neighbor.list() then
15:        my_degree ← my_degree + 1
16:        Neighbor.list().Append(n)
17:    end if
18: end if
19: if (my_time ≥ ST) and (rcv()) then
20:    Neighbor.degree ← rcv().degree
21:    my_state ← ELECTION_DECLARATION
22: end if
23: if (my_time ≥ 2 · ST) and (rcv()) then
24:    if (my_degree > rcv().degree) or (my_degree == rcv().degree) and
   (my_addr > rcv().addr) then
25:        my_role ← CH
26:    else
27:        my_CH_count ← my_CH_count + 1
28:        my_role ← CM
29:    end if
30:    my_state ← CONNECTION_REVELATION
31: end if
32: if (my_time ≥ 3 · ST) and (rcv()) then
33:    for n in Neighbor.list() do
34:        if (n.role == CH) and (my_CH_count ≤ 2) then
35:            my_CH_count ← my_CH_count + 1
36:            my_CH_list ← n
37:        else
38:            my_role ← CB
39:        end if
40:    end for
41: end if
42: my_state ← CONNECTION_DECLARATION
43: if (my_time ≥ 4 · ST) and (rcv()) then
44:    if (my_degree > rcv().degree) or (my_degree == rcv().degree) and
   (my_addr > rcv().addr) then
45:        my_role ← CBH
46:    end if
47:    my_state ← CONSENSUS_CONVERGENCE
48: end if                                              ▷ //End of clustering
```

tion revelation and connection declaration states are shown in lines 32-48 of Algorithm 6 with a transition to the consensus phase after the ST interval.

The clustering phase concludes with the connection declaration state and transitions to the consensus phase upon the expiration of the subsequent ST interval. It is important to note that the consensus phase starts with a new reference time (common across the network) and less contention on the network as message transmission is restricted only to CHs and CBs to finalize the time slots. The CM nodes listen to the information from CH transmissions and update their clocks. All nodes transmit in the discovery phase continuously while only the CH and CB nodes transmit beyond the election phase in a time-slotted manner. Consequently, the contention on the channel as well as the power expenditure of the nodes are minimized. In contrast, most flooding-based protocols experience constant channel contention among nodes leading to significant packet and energy losses.

The topology/status of all nodes (CH/CB/CM) of the clustered architecture remains intact for the consensus phases. However, any network changes (node mobility or unstable communication links) are reflected in the subsequent discovery phase, where different nodes may get elected as CH/CB depending on the change. The discovery phase gets triggered after the configured number of repetitions (MAX_COUNT) to the consensus synchronization phase are complete. The configuration is set based on the frequency of changes in the network, *i.e.,* dynamic networks with higher node failures/mobility have shorter repetitions and vice-versa. New nodes become CMs and listen to the nearest CH for the time information while missing nodes are treated as fail-stop faults and handled according to the steps discussed in Section 5.3. Discussion on the impact of mobility of nodes is presented in a later section.

5.2.2 Consensus phase

The consensus phase is a stable repetitive phase after clustering for maintenance of synchronization and the clustered architecture. There are two states in this phase: consensus convergence and consensus synchronization.

Consensus convergence state. In this state, a pre-configured number of time slots are available for the CH nodes to transmit their messages. The slot of a CH is decided based on the number of its CH neighbors obtained in the connection declaration state. For example, a CH with only one CH neighbor node transmits first, followed by CHs with two neighboring CH nodes and so on. Generally, CH nodes with a lower number of CH neighbors (one or two CH neighbors) tend to be at the edge of the network while CH nodes with a higher number of CH neighbors (two or more CH neighbors) are located towards the center of the network. This fact is exploited by our protocol to find one or more local center (LC) nodes depending on the size of the network that can act as a time source to synchronize different parts of the network, leading to localized time distribution. LCs are CH nodes that have the highest neighboring CH connections and are typically located towards the center of the network. Lines 1-11 describe the consensus convergence state in Algorithm 7.

Algorithm 7 Pseudo-code representing the operation of Consensus phase in C-sync.

```
1: for slot in MAX_SLOTS do
2:     for CH_neighb in CH_neighbors do
3:         if my_CH_degree ≤ CH_neighb then
4:             my_slot ← slot
5:         else if slot == MAX_SLOTS then
6:             my_slot ← MAX_SLOTS
7:         end if
8:         my_state ← CONSENSUS_SYNCHRONIZA- -TION
9:     end for
10: end for
11: my_slot = MAX_SLOTS − my_slot
12: while consensus_count ≤ MAX_COUNT do
13:     for slot in MAX_SLOTS + 1 : 2 · MAX_SLOTS do
14:         if my_slot == slot then
15:             increment(consensus_count)
16:             my_time ← rcv().time
17:             if my_role == CB  or CH then
18:                 broadcast(Msg)
19:             end if
20:         end if
21:     end for
22: end while
23: my_state ← DISCOVERY                                  ▷ //End of consensus
```

The number of time slots is proportional to the number of hops a node at the edge of the network traverses to an LC node. Additionally, in the case of special network topologies like a chain or a ring, where most nodes have the same set of CH neighbors, the configurable number of time slots can be set to limit the maximum number of hops to reach the LC node. If a CB receives a message from a CH, the CB node acknowledges and confirms the CH's time slot (say slot 1). A CB that receives messages with different time slots from neighboring CHs chooses to acknowledge the higher slot number by convention. This is because a CH node with a higher time slot tends to be closer to the central node than its lower counterpart (nodes with lower timeslots transmit first and are located at the end of the network). Similarly, a CH node updates its time slot to a higher value if its neighbor slot is higher than its initial slot (in the case of chain/ring topologies). The transmission continues until all the CH nodes in the neighborhood have a confirmed time slot. Similar to slot selection, if there are multiple nodes with the same number of neighboring CH and time slots, the node with a higher ID is chosen as the LC. This property can be also configured to take the average of the time information received from both nodes. The culminating CHs in different parts of the network are known as the *local centers* of the network. Multiple LCs are found depending on the size of the network and these nodes disperse the time information back to the CH nodes.

Figure 5.3 shows an example of CH nodes in a network of randomly distributed nodes. Note that the CM nodes are not shown for better readability while CB nodes are assumed to be embedded within the links connecting the nodes shown in Figure 5.3. The CH

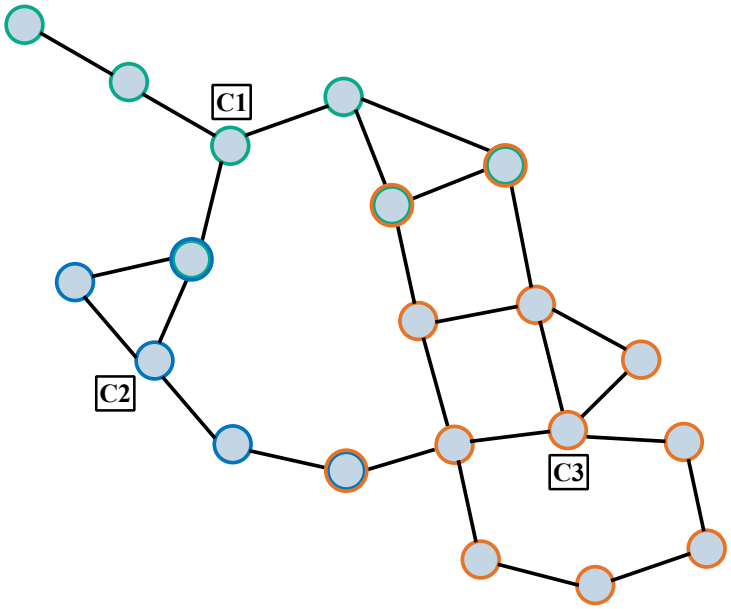


Figure 5.3: The process of finding LCs from the edge CHs and their associated nodes which receive time information.

nodes at the edge of the network with a lower number of CH neighbors start transmission until the LCs - C1, C2 and C3 are found. Further, the LCs transmit the time information back to the CH at the edge of the network in the *consensus synchronization* state. The colors indicate the nodes associated with the corresponding LCs. As seen from the figure, some CH nodes receive time information from two LCs. In this case, such nodes can choose to associate themselves with either of the LCs. All CM nodes during this state are in sleep mode during the convergence and wake up for 1 slot to receive their slot numbers for the synchronization state to receive the time information. An example with 10 slots for the convergence and synchronization state is shown in Figure 5.4.

A notable caveat is that the minimization of hops is adaptive to the network and the topology, *i.e.,* a network could have multiple LCs (within a distance of 1-2 hops) or a single LC (configured maximum hops). This novel method of limiting the number of hops to the time source achieves a simplistic solution eliminating the requirement of any additional hardware/timing adjustment. Although existing solutions are able to limit the error significantly, it is functional only up to a certain number of hops and the problem repeats upon further scaling the network.

Consensus synchronization state. In this state, LC transmits time information to the CH nodes and further to the CM nodes of their respective clusters. The time slots for this state are the modulo time slot proportionate to the pre-configured time slots, *i.e.,* if a CH node transmitted at slot 4 in a 10-time slot window during convergence, it receives its time information at slot 6 (10-4) during synchronization. CM nodes are awake to receive their slot numbers from CH (same as their associated CH's slot) and

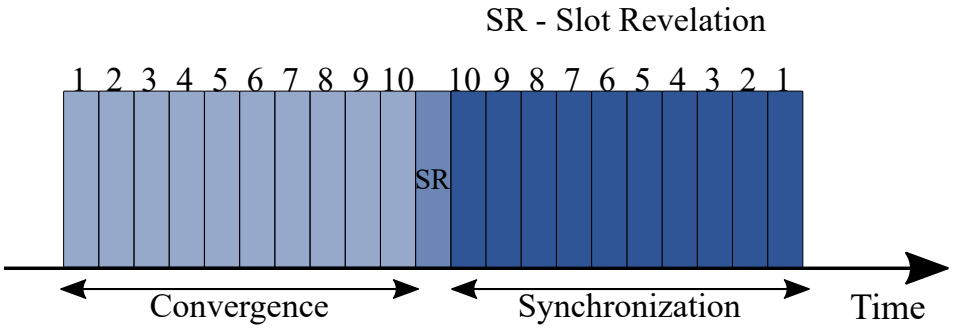


Figure 5.4: Example of consensus states with 10 slots.

go into sleep mode. Since the time information is passed from the local center, the clock rate and offset are updated relative to the LC. This operation is shown in lines 12-23 of Algorithm 7.

Synchronization errors accumulate at every node starting from the LC until the CM nodes along the path. In the case of an ideal clock, offset compensation would be sufficient to synchronize every node to the LC by computing the difference. As practical clocks have variations in both offset and drift, both parameters need to be compensated. The logical clock rate of a receiving node (r), from Equation (5.1), is defined as the ratio of a logical clock (global clock value) of a node to its hardware clock after offset compensation and is given by:

$$l_r(t) = L_r(t)/h_r(t). \tag{5.4}$$

Since the rate is dependent on the hardware clocks of each node along the multi-hop path, it is important to adjust the rate only to the logical clock of the LC as the reference clock. The relative clock rate of receiving node (r) relative to the sending node (s) is the ratio of the logical clock of the sender ($L_s(t)$) to the hardware clock of the receiver ($h_r(t)$), given as:

$$l_{rs}(t) = \frac{L_s(t)}{h_r(t)}. \tag{5.5}$$

If a node is directly connected to the LC, the relative rate would be sufficient to compute the logical clock. However, for any non-direct neighboring node to the LC, this relationship causes dependency of hardware clocks of all the nodes on the path from its LC. Hence, for non-direct neighbors, we compute the logical clock rate with reference to LC (l_{LC}) as the ratio of the relative rate to the current clock rate of the receiver:

$$l_{LC} = l_{rs}(t)/l_r(t). \tag{5.6}$$

This new rate is computed and transmitted at every node along the path to the edge of the network. Each node can synchronize to the LC by utilizing the product of the relative rate to the sender and the l_{LC}.

CH nodes turn on only in their respective slots while CB nodes remain active for two slots to receive and send information to their CH neighbors respectively. The clock rates

of two LCs are averaged to establish a uniform clock synchronization across the network if a CH node in any path is also an LC. The nodes move into an idle phase after synchronization where they are in sleep mode and no messages are being exchanged. C-sync switches to consensus synchronization periodically for LC to distribute time information for maintaining synchronization. The idle phase conserves power and prevents the constant exchange of messages to maintain synchronization. They re-synchronize when they wake up after the sleep cycle to correct the clocks that are drifted apart during the idle phase.

C-sync also switches back to the discovery phase after a few consensus phases to accommodate ad-hoc networks where new nodes may join in or existing nodes may be removed (due to energy exhaustion, movement or faulty nodes) through a periodic switch back to the discovery state to create more efficient clusters. Due to the periodic switch to the discovery phase, C-sync can adapt well to dynamic networks with regular failure or mobility of the nodes. However, the switching frequency to discovery phase must be carefully configured as to minimize energy expenditure. In stable networks with a low probability of movement/faults, the number of consensus phase repetitions can be configured to a large number to prevent unnecessary re-clustering. This process allows the protocol to make the cluster structure more efficient when there are network changes.

C-sync overhead. Existing synchronization solutions have an overhead to cover the entire network diameter with a lack of backup mechanisms to handle a dynamically changing network. On the contrary, the diameter in C-sync ranges from a single cluster to distance (hops) to the LC, yielding a much lower overhead. C-sync employs re-clustering to ensure an efficient clustered architecture and has a fault detection and correction mechanism in place to handle changes in the network. Additionally, since the protocol starts with a completely decentralized network, the computational complexity of the protocol is $O(n)$, where n is the number of clusters.

Lemma 1. *The maximum synchronization error between any node to its nearest LC is a parameterized value.*

The synchronization error in most time synchronization protocols is dependent on the propagation time, frequency of messages exchanged and the number of hops required to communicate. Due to MAC-layer time-stamping, the propagation time can be safely ignored assuming no channel contention and interference[21]. This is because the propagation delay roughly amounts to 0.3 µs for 100m distance between the nodes while the resolution of hardware timer is about 1.9 µs. It is important to note that the computation applies to the consensus synchronization state. If the delay between two consecutive messages exchanged is τ and the minimum achievable synchronization error for an ideal "zero"-delay is δ, the accumulated error due to delay in message exchange is $\tau \cdot \delta$. In C-sync, τ is the idle-time delay between two synchronization messages, and the maximum number of hops to an LC is represented by η. As the synchronization error increases at each hop, the total error from any node to its LC is given by $\eta \cdot \tau \cdot \delta$. This configurable parameterized limit restricts the synchronization error for any node in the network to its LC. □

5.3 Resilience in C-sync

As stated earlier, resilience to faults is one of the important features of C-sync. In this section, we use the categories discussed in Chapter 2 and list the fault handling-specific assumptions made by C-sync. Further, we prove by induction that any fault in our described categories can be handled by C-sync if the assumptions are met and discuss the fault detection and correction process.

Fault model for byzantine faults. As discussed in Chapter 2, the fault models in IoT networks range from a simple fail-stop fault to a byzantine fault. In this chapter, we consider all the faults types discussed in Chapter 2. With a fail-stop fault, the node is non-responsive due to battery exhaustion, hardware/software damage and/or environmental factors, etc. We also consider byzantine faulty nodes where the nodes could behave erratically making it difficult and expensive in terms of communication to detect them. The subset of byzantine faults considered in this chapter includes spikes, outliers, stuck-at and intermittent communication faults [35, 36, 37] as discussed in Chapter 2. Spike fault is a sudden surge in reported values that may or may not return subsequently return to normal values. When the reported values are beyond the boundary of the expected values, the resulting fault is an outlier. If reported values are constantly stuck at the same value, then a stuck-at fault is observed. It is important to note that the stuck-at faults can only be identified when the reported value is either an outlier or a spike and remains stuck at those values. With intermittent faults, the message transmission from a node is sporadic with periods of inactivity.

Typically, in cluster-based network architectures, the faults described above can be translated as selective forwarding, discovery flooding and altered information [37]. Selective forwarding is prevalent in multi-hop networks where faulty nodes selectively forward some messages while dropping the other messages. Fail-stop faults and intermittent communication mimic a potential temporary loss of communication resulting in selective forwarding. Additionally, nodes may send sudden variation (spikes, outliers) in the information due to a fault (E.g. False perception of the environment, routing changes, etc.) leading to altered information. A threshold on the acceptable range for the received data can prevent the altered information faults such as spikes and outliers. A node with a faulty radio could end up in a high transmit power resulting in discovery floods (also called HELLO floods) and consequently could end up with more neighbors than the normal range, leading to a higher degree. The high-powered transmission could lead to the false election of these nodes as one of the critical routing nodes (CB or CH). For discovery floods, it is important to verify the bi-directionality of the links between the nodes, *i.e.,* to verify that the link has the same properties in both directions. The above faults are the observable faults through radio communication and constitute a subset of the generic byzantine faults. Additionally, we assume that the faulty nodes can cooperate with each other independent of their node type. The impact of the faults observed in both phases of C-sync described in Section 5.2 is studied.

Impact of faults. As seen in DeCoRIC [135], the hard faults can be handled within a bounded time even with the absence of synchronization. With C-sync, it is expected that the CH and CBH nodes transmit at a specific time slot and the impact is more pronounced for CH and CB nodes. During the clustering phase, a faulty node tries to be a routing node (such as a CH or CB node) as common nodes cannot influence the messages beyond their own cluster. A faulty node can report an erroneous degree and/or use discovery flooding with a high-powered (damaged) radio to become a CH/CB node during the clustering phase. When a node becomes a routing node (such as a CH or CB node), it could have a wider impact during the consensus phase transmitting erroneous time information across clusters. Since CB plays a critical role in the fault detection and correction process, an additional authentication using AES cipher is used with a combination of communicating CH nodes' ID as the key to prevent non-neighboring nodes of CHs from getting promoted to a CB. Replicating a MAC address of 8 bytes in the ID through brute-force is a highly difficult and energy-intensive task for a resource-constrained node. Encryption provides an additional layer of defense as it is also assumed that nodes cannot change their addresses or the reference address. The AES cipher is used atop the existing message encryption mechanism of IEEE 802.15.4 [3] to authenticate the CB nodes and verify the bi-directionality of the links [37]. Authentication of CB nodes allow for a simple majority consensus (agreement) among the correct nodes [136]. The Byzantine consensus (agreement) mechanism can detect and correct all the other faulty nodes (CH/CB/CM).

The byzantine consensus (*i.e.,* fault handling) mechanism is described in detail below. Prior to that, the assumptions and broadcast primitive required for byzantine consensus are also discussed.

5.3.1 Network assumptions to achieve byzantine fault resilience

Let n_i denote the number of neighbors of any node i in a cluster within its communication range, $\mathrm{n_{CB}}$ denotes the set of CB nodes between two clusters with cluster heads CH1 and CH2 such that CH1, n_{CB} and CH2 are at an increasing number of hops from the local center respectively. The assumptions and broadcast primitives required for byzantine consensus are:

Assumption 1. *No node can fake its address or the reference address in a message as it is a hardware-based MAC address.*

Assumption 2. *Every node in a cluster has at least* $\lfloor \frac{n_i}{2} \rfloor + 1$ *neighbors that are fault-free.*

Assumption 3. *There are at least* $\lfloor \frac{n_{CB}}{2} \rfloor + 1$ *CB nodes that are fault-free between any two clusters. In the event of a fault, every node expects* $\frac{n_i}{2} + 1$ *correct messages and re-transmits any received messages till the expected number of messages are met.*

Assumption 4. *There are at least two clusters in the network without any network partitions/isolated clusters.*

Atomic broadcasts

Typical byzantine agreements require significant energy to perform computations and communication [38]. Sensor nodes are resource-constrained and require an energy-efficient way to achieve a byzantine agreement. Atomic broadcasts were introduced in [137] to achieve byzantine consensus if they meet the following criteria:

- Every message from a correct sender is received by all correct receivers within a time-bound.
- Every message is received by the correct receiver in the same order as it was sent by the correct sender.

The byzantine agreement is concluded if all the correct nodes have the correct information that was propagated.

In C-sync, atomic broadcasts are initiated by the CB nodes if the CBH or CH nodes are non-responsive or send incorrect information. To understand further, we take a look at both types of faults and present the fault handling mechanisms.

5.3.2 Fault detection and correction

As discussed in Chapter 3, DeCoRIC handles hard faults within a bounded time even in the absence of synchronization [135]. In most IoT networks, due to high density of nodes, the minimum number of nodes based on the assumptions can be achieved. With C-sync, it is expected that the CH and CBH nodes' synchronization messages are transmitted at a scheduled time slot. If there is a missing broadcast due to a damaged node or selective forwarding, it is treated as a fail-stop fault. Byzantine faults arise when the time information is modified to destabilize the cluster and network. Both the faults are handled through byzantine consensus among nodes of the cluster.

All CBH and CB nodes schedule a byzantine consensus message at a delay of two message transmissions time after the transmission of the synchronization message and monitor the information within the cluster to verify its authenticity. Two transmission times account for message transmission from CBH to CH and, further, from the CH to all the cluster nodes. CB nodes monitor the information shared by the CBH and CH nodes to verify the authenticity of the information shared. The CB nodes drop the scheduled message upon correct information from CBH and CH nodes in their respective time slots. Alternatively, if the shared time information exceeds a configurable threshold (synchronization error $>$ threshold) or a missing/failed transmission causes a timeout, a fault is recorded and atomic broadcasts for byzantine agreement are initiated immediately. Note that a consensus message can be configured to trigger after a certain number of repetitive misses/violations. The detected faulty node is added to a blacklist where messages originating from the blacklisted nodes are ignored by the other nodes and are excluded from forming new clusters. The blacklisted nodes are excluded when protocol moves into the discovery phase and new clusters are formed without the blacklisted nodes. Upon receiving the atomic broadcast, all the CM nodes and/or CH nodes (non-faulty) re-transmit this information till every node in the cluster receives at least

$\lfloor \frac{n_i}{2} \rfloor + 1$ correct messages so that all the nodes in the cluster receive the information. The above condition is both *necessary* and *sufficient* to achieve a successful byzantine consensus (agreement).

A faulty CM node can only assert its influence by sending a false byzantine consensus message. However, this message is not replicated by all of the other CM nodes as it was not sent by any of the CB nodes directly/indirectly (as reference). Thus, the minimum number of messages will not be received by any of the other nodes of the cluster, leading to the failure of atomic broadcasts. The clustered architecture and byzantine consensus mechanism established in C-sync can handle all the aforementioned fault types and can further be extended to handle other fault types and even certain attacks. The extensions are left as a future work of the presented algorithm.

The flooding of messages within the cluster leads to a temporary increase in power consumption. However, it is a small price to pay to contain faults within a cluster and achieve resilience against faulty nodes. It is important to note that the byzantine consensus completes within the same time slot when the time information has to be distributed and hence, does not impact accuracy.

Theorem 2. *Any faulty node in the network can be detected and corrected in C-sync if the stated assumptions are satisfied.*

Proof. We prove the theorem using the principle of induction. The proof is provided with reference to a single cluster as a representative case consisting of a cluster head CH that connects to other clusters through CB nodes and the associated CM nodes such that every node meets Assumption 2. The cluster is connected to other clusters through CB nodes. Let CB1 and CB2 denote the set of CB nodes of CH and its neighboring clusters, with CB1 located closer to the LC and CB2 located farther away from LC, such that the information chain traverses from LC to CB1 to CH to CB2. Multiple nodes are present within the sets CB1 and CB2 to ensure that Assumption 3 is met. Without loss of generality, the same proof applies to every cluster in the network independently. Also, the proof for the CB1 set applies to the CB2 set as well.

For the base case, let us assume there is only one fault in the cluster. If CH is the faulty node, the information from CB1 is either dropped or modified before distributing it to CB2 and its CM nodes. CB1 nodes including CB1H (CBH in the set of CB1) send a byzantine consensus message immediately (modified time information) or as scheduled (selective forwarding) such that it gets re-transmitted across the cluster upon reception. Based on Assumptions 2 and 3, the fault-free $\lfloor \frac{n_i}{2} \rfloor + 1$ neighbors propagate the correct information from the received byzantine consensus messages to reach an agreement. Hence, the faulty information gets detected and corrected. Similarly, if the CB1H node or one of the CB1 nodes skip sending the synchronization message or send faulty time information, the remaining CB1 nodes initiate the byzantine consensus. A faulty CM node can directly trigger a byzantine consensus message with incorrect time information. However, since the CB1H address cannot be used as a reference, the fault-free neighbors do not re-transmit this message. By Assumption 2, byzantine consensus for the faulty CM node will not be reached.

Let us assume that the C-sync protocol can detect and correct up to k faults that satisfy assumptions 1, 2 and 3, *i.e.,* $k < \frac{n_i}{2}$ for all the cluster nodes including CH and $k < \frac{n_{CB1}}{2}$ for the set CB1.

Consider $k + 1$ faults that satisfy the assumptions 1, 2 and 3 and the assumption of k faults as stated above. If the additional faulty node in the $k + 1$ faults is a CM node and assumption 1 holds, this CM node can trigger a byzantine consensus message without CBH or any of CB1 as reference nodes. Additionally, with assumption 2, we can infer that $(k + 1) < \frac{n_i}{2}$. Thus, the fault will not be propagated and consensus on faulty information will not be reached *i.e.,* correct nodes are not impacted. With CH as the faulty node, CB1 nodes initiate byzantine consensus messages immediately if the error from the CH node exceeds the threshold. Since $(k + 1) < \frac{n_{CB1}}{2}$ and $(k + 1) < \frac{n_i}{2}$ hold, then the messages with correct information from the $\frac{n_i}{2} + 1$ nodes (CB1 and CM) of the cluster dominate the faulty byzantine messages $(k + 1)$ leading to successful detection and correction of the fault. Lastly, if either CB1H or any of the other CB1 nodes is the additional faulty node, and $(k+1) < \frac{n_{CB1}}{2}$ holds, the messages from the $k+1$ nodes are not sufficient to reach the byzantine agreement. Thus, the correct CB nodes are able to disseminate the information within the cluster.

Theorem 2 holds for any k such that assumptions 1, 2 and 3 are satisfied. Hence, with the C-sync protocol, wireless networks can achieve resilience to faults. □

Discussion

An exception to the fault handling in C-sync is the case of node mobility during message transmission. A mobile node could initiate a transition from one cluster to another cluster during the synchronization message transmission (consensus synchronization state), mimicking a fail-stop fault/selective forwarding. Consequently, the node gets blacklisted although movement was a legitimate action. However, there is no impact if the mobile node never returns to the original cluster. This exception can be addressed through an exchange of membership information in addition to time information among different CH nodes of the network in the consensus convergence state and is beyond the scope of this thesis. Additionally, small networks that result in a single cluster without a CB are also an exception to the fault handling mechanism of C-sync. To handle faults in this scenario, the CM nodes would be awake for a longer duration to initiate a consensus flooding (if required) resulting in a power efficiency vs fault tolerance trade-off.

5.4 Experiments and results

To test the performance of the C-sync protocol, we introduce a fault in a CH node of a simple representative network to show that the fault detection, correction and containment within the cluster are achieved. Further, we conduct experiments to compare the power consumption to achieve synchronization and the resulting accuracy of synchronization. Finally, we demonstrate through a chain topology that the synchronization error for any node to its nearest time source is restricted. In Section 5.4.1, we explain the details of the experimental setup used to conduct our experiments. We discuss the

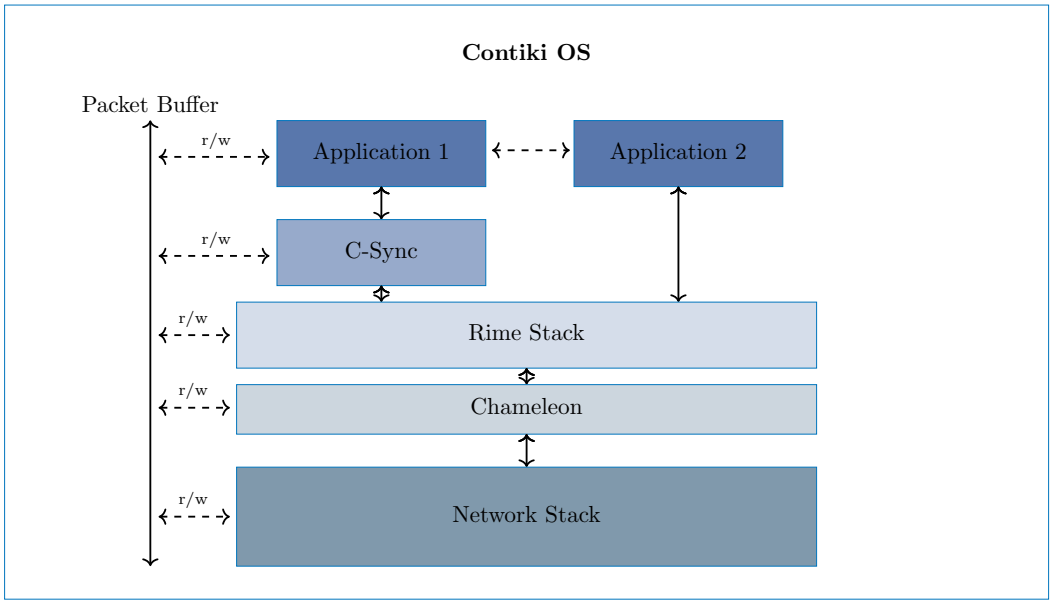


Figure 5.5: Software architecture of Contiki OS integrated with C-sync.

results from our experiments for energy efficiency, synchronization error propagation, and fault detection and correction in Sections 5.4.3, 5.4.4 and 5.4.2, respectively.

5.4.1 Experimental setup

The C-sync protocol was implemented on the Contiki 3.0 operating system [138], with Tmote Sky [4] boards utilizing the IEEE 802.15.4 communication standard. While nodes with the same capabilities are used for testing purposes, C-sync can be used in networks having nodes with heterogeneous capabilities. Tmote Sky consists of MSP430 [139] micro-controller and CC2420 [140] radio communication chip from Texas Instruments. The experiments were conducted on the Indriya [31] testbed at the National University of Singapore, where there are over 50 Tmote Skys deployed on different floors of a building. The implementation has been made available publicly for use by the community.

The software architecture of C-sync is shown in Figure 5.5. Starting from the lowest layer, the network stack maps the bottom three layers of the Open Systems Interconnection (OSI) protocol stack whose output is passed to the upper layers. The network stack handles the radio communication including physical layer control, link-layer security features such as AES, CSMA-CA, and MAC-layer features such as MAC-layer timestamping. It also provides link-layer security features among which AES encryption is used in the connection declaration phase for the CB election. The output of the network layer is the received packet buffer that is passed to the upper layers. Chameleon is a header transformation layer that adds or removes the header component from the packet buffer.

Rime stack is a versatile layer that provides various implementations of communication primitives such as periodic broadcasts, etc. This layer handles the packet buffer queuing

in case the channel is busy due to CSMA-CA. Rime also offers a variant of the broadcast primitive called polite announcement that reduces the messages within the radio range of a node by monitoring the channel for any duplicate messages. If a duplicate message is received, the scheduled transmission packet is dropped; and the transmission continues once the channel is free or if a different message is received.

C-sync runs concurrently over the Rime stack, i.e., C-sync uses the communication stack when other applications are idle and vice-versa. Applications that require energy-efficient and resilient time synchronization for their application may use C-sync, while other applications can directly communicate using the Rime stack. During the idle phase of C-sync, the applications directly take over the network stack till the next scheduled consensus synchronization/discovery state using interrupts and function callbacks with time synchronization as a feature that facilitates applications. After C-sync completes the consensus synchronization phase, the idle phase takes over. It is important to note that all the layers operate on the same packet buffer and it is passed among the layers every time it is populated or depopulated depending on the direction of the packet. For comparison purposes, we have also implemented the Gradient Time Synchronization Protocol (GTSP) [15] on Contiki. GTSP was chosen for comparison as the representative protocol among the class of decentralized solutions since it forms the basis of averaging and consensus features used in the other recent solutions. GTSP is also the only decentralized synchronization protocol that has shown feasibility for hardware testing for generic wireless sensor networks.

Rtimer. A software wrapper is implemented to sample the clock from the registers at regular intervals such that a stable clock output is obtained. As seen in Chapter 2, two clock sources can be combined to produce a low-resolution stable output from the timers. In our implementation, we configure the crystal oscillator frequency to be 512 Hz using the clock divider registers. With Timer A sampling the crystal oscillator and Timer B sampling the DCO, at 512 Hz, every 4 samples of crystal produces 4096 ticks of DCO clock.

The deviation of the DCO from the ideal value of 4096 ticks is captured as the clock drift. To get a high-resolution stable clock, the crystal oscillator is used to sample the DCO at regular intervals and obtain the DCO drift factor as shown in Figure 5.6. Every 4 ticks of timer-A correspond to 4096 ticks of timer-B: this definition is used to compute the correct value of timer-B independent of any variation of the DCO as seen in Figure 5.6(a). Ideally, timer-B has to follow the timer-A value without any variation. Ideal clock and DCO variations cause timer-B variations as shown in Figure 5.6 (b). The drift of DCO is computed as the ratio of the estimated value tb_e over the actual value of timer-B tb_a as:

$$\text{drift} = tb_e/tb_a \tag{5.7}$$

This drift factor is multiplied with the current timer-B value to obtain the exact hardware clock value. With the example shown in Figure 5.6, the drift factor is $4096/3904 = 1.04918$. The product of drift with clock value would provide the exact clock value as $1.04918 * 3904 = 4096$. This provides a drift compensated stable DCO output clock from Timer B at a resolution of 2μ s.

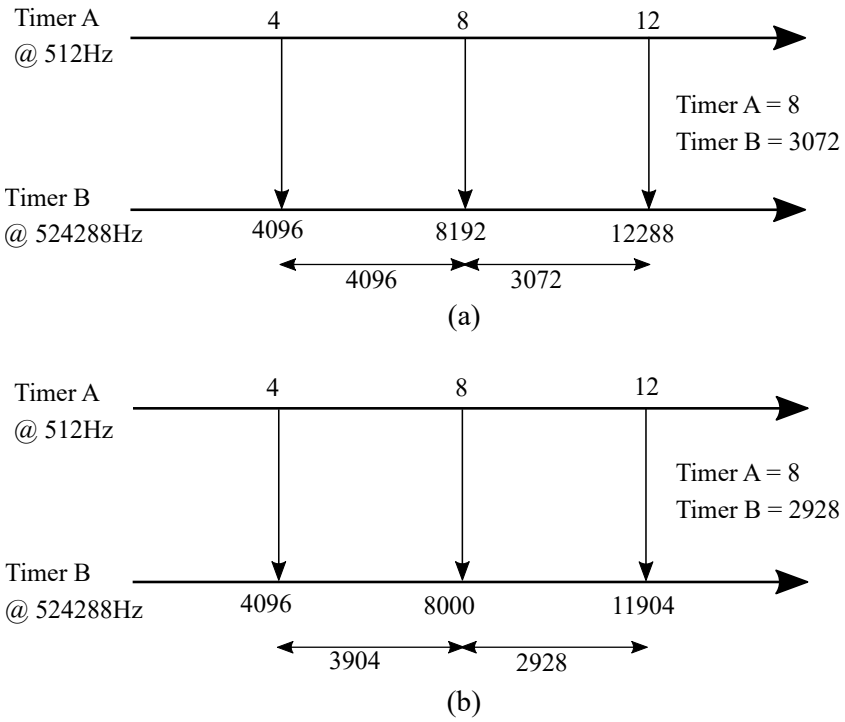


Figure 5.6: DCO time drift compensation using the stable low-frequency crystal oscillator.

Since Timer A operates at 512 Hz, the register overflows every 128 seconds incrementing a variable *coarse_count*. Similarly, when Timer B overflows, a variable called *fine_count* gets incremented. Hence, a timestamp reads 4 values - coarse_count, Timer A, fine_count and Timer B. As the variables are 32-bits in size, the entire timestamp overflows roughly once in 17,432 years. Given the long duration, the rtimer setup is also UTC compatible.

Boot time. The initial boot time of the C-sync to switch from discovery state and achieve loose synchronization is dependent on the density of the network as denser networks take longer to converge due to a larger set of unsynchronized neighbors. Empirically, in our setup, we found the initial transition to the election phase was roughly 10x the ST interval. Hence, the time to complete the consensus phase from the start of the discovery phase is 10+4 ST intervals.

5.4.2 Fault detection and correction

The fault detection and correction steps in C-sync are tested by introducing a fault into one of the nodes. Since the fault handling mechanism is similar across all node types (CH, CB or CM), fault at one type of node is representative of all the other types of nodes. With the CB nodes monitoring the flow and correctness of the information, the fault detection at either of three node types follows a similar pattern with exceptions discussed

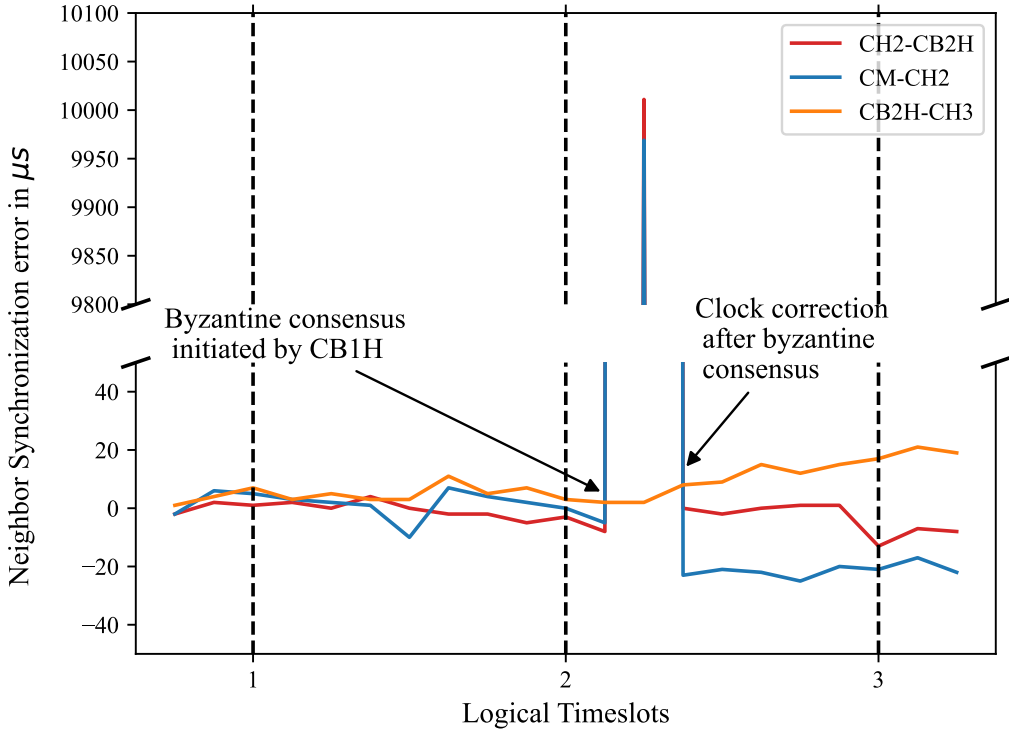


Figure 5.7: The synchronization error of different nodes within a cluster having a faulty CH. The error remains within one logical time slot without any impact on network synchronization.

in Section 5.3.2. Additionally, the fault handling in a single cluster is representative of the fault handling in the entire network as each cluster counters the fault the same way as any other cluster.

We consider an example of three clusters with CH nodes CH1, CH2 and CH3 connected by the bridge node CB1H and CB2H similar to Figure 5.1. CH1, CH2 and CH3 are cluster heads of the clusters and the information flow goes in the order of CH1, CB1H, CH2, CB2H and CH3 from left to right. Let us suppose CH2 is a faulty node in the network. Since only nodes of the CH2 cluster are impacted by the fault, we consider only the common nodes in the CH2 cluster denoted as CM. The threshold for an error to initiate byzantine consensus is set to $500\mu s$. Synchronization error is measured as the difference between the clock ticks in the logical clocks of the receiving node and the neighbor node. CH2 introduces a synchronization error of $10000\mu s$ in its time slot (slot 2) after receiving a message from LC, as seen in Figure 5.7. The CB2H node tries to synchronize to this initially resulting in a high error. The high error is shown with a broken axis on the plot where both CB and CM are impacted by faulty time information. The vertical lines on the plot for CH2-CB2H and CM-CH2 converge to the point in the upper half of the plot. However, byzantine consensus messages with the correct clock value are flooded within the cluster by the CB1H node of the first cluster as shown in Figure 5.7. Both the CM and CBH nodes reset their clocks to the new value

in the received information in the byzantine consensus messages. It is important to note that the consensus completes within the same slot 2. The logical timeslot represents the duration of a synchronization slot, *i.e.,* a combination of consensus synchronization state and the idle state.

CB2H transmits the updated correct clock information to CH3 in the next time slot. As seen from Figure 5.7, the neighboring cluster head CH3 is not impacted by the fault in the cluster with CH2. Hence, the fault is contained within a single cluster and recovers with byzantine consensus as expected from the discussion in Section 5.3.2. It is important to note that a fault (if any) in the clustering phase, gets detected in the consensus phase as the errors and fault tolerance are a consequence of nodes elected in the clustering phase.

5.4.3 Energy efficiency for fault-tolerance

To demonstrate the efficient operation of C-sync, we conducted experiments to measure the synchronization error and the power consumed for achieving neighbor synchronization as shown in Figure 5.8. Neighbor nodes are a part of the clustered architecture, including, but not limited to communication between CM-CH, CB-CH and CH-CB-CH. A scatter plot of average neighbor synchronization error in μs on the y-axis against average power consumption of each node in mW on the x-axis after the protocols move to the idle phase is plotted. Each point on the plot represents an averaged value of power and synchronization error for each node in the network over the entire duration of the experiment. Measured values of synchronization error are plotted without taking an absolute value, *i.e.,* the offset can be both positive or negative.

Although the error and the power are not correlated, they provide an intuition on the synchronization protocol performance to achieve low synchronization error and power consumption. The power and synchronization error is measured after the protocol moves onto the idle phase. Ideally, a probe is needed to measure the offset physically at each node. Alternatively, in Figure 5.8, synchronization errors of nodes with respect to their neighboring nodes are measured through a message exchange in the idle phase to only compute the offset with no rate/offset compensation. Hence, no power expenditure is recorded for these messages. Similar to C-sync, the GTSP idle phase begins at the end of the discovery phase since GTSP does not employ a state machine in its protocol.

The scatter plot consists of tests conducted in three different topologies formed by utilizing different configurations of the testbed. The average synchronization error across all nodes in the network is not shown in Figure 5.8 for better readability. The summary of the results of average neighbor synchronization error (offset) and average network power consumption (power) with corresponding standard deviations is shown in Table 5.2.

The dense distribution of nodes is formed by using the different pockets of nodes closely grouped together on each floor of the building (denoted as dense n/w in the figure). GTSP performs best in a dense environment due to the closeness of the nodes and, hence, converges quickly with a synchronization error of $7.05\mu s$. However, due to the continuous exchange of messages, there is a high power consumption of $0.98mW$. C-sync forms larger clusters with more CM nodes in a denser environment. Hence, more nodes remain in sleep mode while achieving a similar synchronization error of $7.52\mu s$ and

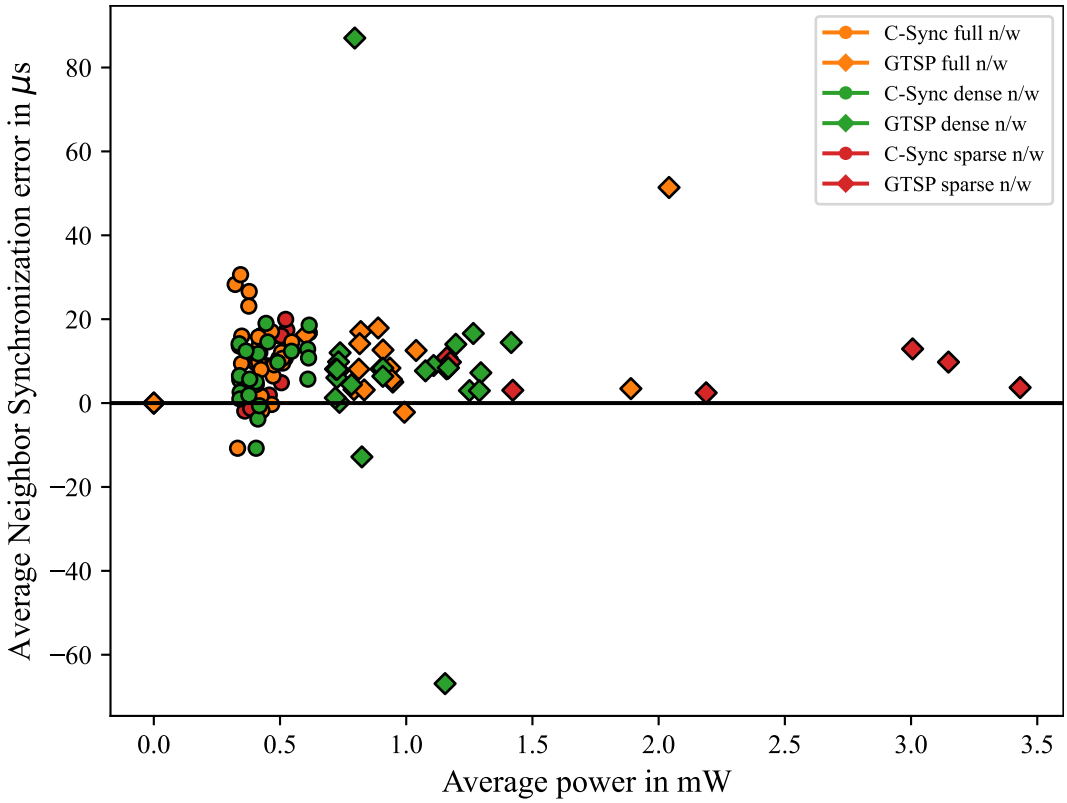


Figure 5.8: Scatter plot of average neighbor synchronization error and the average power consumption for each node over different topologies that are densely distributed, sparsely distributed and using the entire testbed compared for C-sync and GTSP.

Table 5.2: Results of average neighbor synchronization error and power consumption measured (with associated standard deviation) over 30 minutes across different topologies.

	Dense Network	Sparse Network	Full Network
GTSP sync. error (μs)	7.05 (24.51)	7.67 (3.73)	10.13 (11.64)
C-sync sync. error (μs)	7.52 (7.22)	10.05 (6.78)	12.24 (9.35)
GTSP Power (mW)	0.98 (0.24)	1.98 (0.97)	0.89 (0.48)
C-sync Power (mW)	0.43 (0.1)	0.48 (0.07)	0.43 (0.06)

low power consumption of $0.43mW$, having a reduction of power by 56.12% compared to GTSP.

The sparse distribution of nodes is formed by utilizing a few nodes from each floor to communicate with each other (denoted as sparse n/w in the figure). Due to the sparse distribution, the convergence speed of the 1-hop synchronization algorithm is impacted while multi-hop synchronization performs better [141]. Hence, due to the neighbor-only synchronization, GTSP incurs a longer convergence time with a synchronization error of $7.67\mu s$ and power consumption of $1.98mW$. However, C-sync synchronizes to the closest neighbors to form smaller clusters while letting farther nodes become CB nodes to other similar clusters. Hence, there is a multi-hop synchronization across multiple clusters yielding a synchronization error of $10.05\mu s$ and power consumption of $0.48mW$ which is 75.75% lower than GTSP. This way, there is a faster convergence, yielding significant power reduction.

Combining both the above environments, utilizing all the nodes of the testbed provides us the results for a full network (denoted as full n/w in the figure). GTSP synchronizes to the mixed environment yielding an average synchronization error of $10.13\mu s$ and consuming $0.89mW$. C-sync forms multiple clusters with a mixture of large and small clusters connected via cluster bridges. C-sync achieves an average power consumption of $0.43mW$ which is roughly 51.68% of the power consumption of GTSP in the full network topology. The synchronization error is similar to GTSP with C-sync having an average synchronization error of $10.13\mu s$. Both protocols take a longer time than the dense and sparse configurations to achieve synchronization due to the higher number of nodes in the network.

The difference in average synchronization error between C-sync and GTSP across various topologies is at a maximum of about $2.4\mu s$. This is equivalent to one tick of the clock at a resolution of $1.9\mu s$. This difference can be attributed to the measurement error and hence, can be concluded that the accuracy of both the protocols are similar.

Since a small change in ticks could result in a larger change in the synchronization error, we observe that the standard deviation is higher during the measurement of the synchronization error. C-sync consistently achieves the roughly same deviation in synchronization error, whereas GTSP experiences a larger deviation depending on the topology. The fixed set of states and the leader-follower synchronization in C-sync ensures that the protocol converges and moves into the idle state faster, leading to lower variation in power consumption.

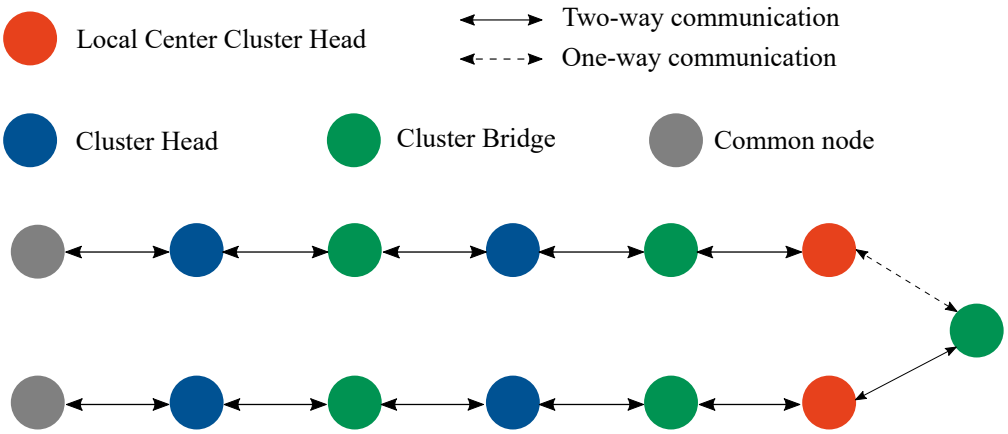


Figure 5.9: Chain of Cluster Heads and Cluster Bridge ignoring the Common nodes to demonstrate a multi-hop network.

5.4.4 Synchronization error to local center

As described in the protocol, C-sync is able to restrain the distance of any node to the time source within a pre-defined parametric threshold. In order to demonstrate this experimentally, we constructed a chain topology to emulate a multi-hop network as shown in Figure 5.9. Ignoring all the CM nodes except for the ones on the corner CH nodes, there are 13 nodes in the chain.

Letting C-sync run on every node, a chain of CH and CB was established, with each node (except the CM nodes) connected to two neighbors. The synchronization error from the LCs to each of the nodes associated with the LCs is plotted over the number of hops it receives the time information. In the consensus convergence phase, starting with the CH at both ends, every node starts to transmit and increment its slots. Once the slot count reaches the maximum number of slots or reaches a CH node with the highest neighbors in the neighbor, the Local Center (LC) is found. LC is responsible to distribute time information to its neighborhood and the CH nodes receive this information in the reverse order of convergence. As seen in the example, LC is reached from both ends and the edge nodes synchronize to the time information sent by their associated LC over multiple hops.

The CB node connecting both LC receives information from both but synchronizes to only one of them based on the address of the LC (since both LC have the same hop count). This can be configured to average the time information and further transmitting it for synchronization among LCs when LCs have different slots of transmission. Generally, CB nodes synchronize to one of the CH neighbors and pass the information to the other CH neighbor. Since both LCs have the same hop count in our example, both LCs are active in the same time slot. Hence, the CB information is not received by the LC in the next time slot. The entire network gets re-synchronized at the next discovery phase.

For our example network, the maximum number of hops from any node to its LC was observed to be five. Each slot in the synchronization phase is $300ms$ and the idle phase is configured to be ten such slots. Computing the periodicity of messages received where

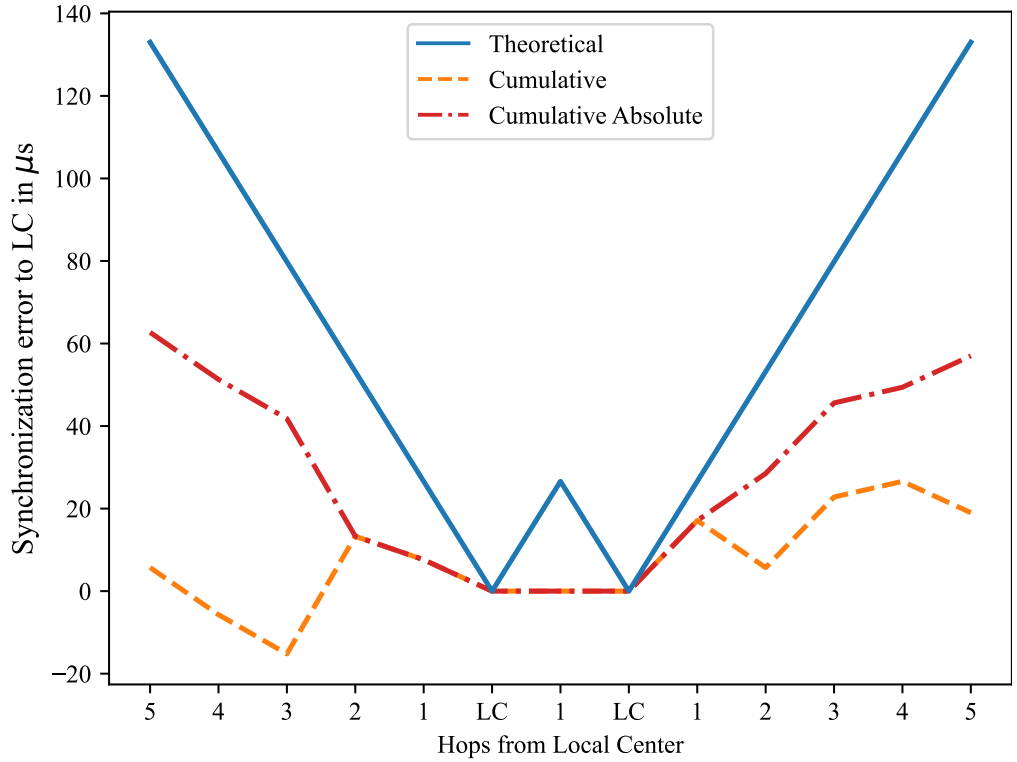


Figure 5.10: Synchronization error is bounded by the established the maximum number of hops from the local center.

every node is active only for 1 slot, each node receives a synchronization message once every $10+(5-1)$ slots, *i.e.,* a periodicity of 14 slots. The ideal minimum synchronization error to the LC in the case of C-sync is the duration of 1 instruction (1 tick) of recording MAC-layer timestamping. Based on the resolution of the clock, it is approximately $1.9\mu s$. Using the information of maximum hop count and the minimum synchronization error, the worst-case calculated synchronization error to the LC can be calculated from our previous definition of $\eta \cdot \tau \cdot \delta$ as $(1.9 \cdot 14 \cdot 5) = 133\mu s$. However, this limit reduces if the number of hops of a node is lower than five. The experimental measurement for the synchronization error to the LC over multiple hops is plotted against the estimated value as shown in Figure 5.10.

At the LC, the error is zero since its own clock is the reference clock. As seen from the plot, both the cumulative error and the absolute cumulative error (converting negative offset to positive) were measured and found to stay below the computed theoretical value.

5.5 Summary

In this chapter, we presented C-sync, a clustering-based decentralized time synchronization protocol that is both resilient to faults and energy-efficient. C-sync maintains

suitable accuracy and uses the clustered architecture to enable more nodes of the network to remain in sleep mode. The clustered architecture of C-sync paves way for resilient design and adaptability of real-time applications on decentralized networks. C-sync can remain in a consensus-idle phase loop as long as there are no network changes, significantly reducing the messages exchanged to maintain synchronization in the network. The implementation has been done on Contiki and a hardware testbed.

The fault handling mechanism with byzantine consensus was described and demonstrated experimentally by introducing a fault in a simple network topology that can be scaled. We illustrated through experiments that C-sync achieves significantly lower power consumption compared to GTSP while attaining similar accuracy. Additionally, the concept of local centers was introduced and their role in restricting the synchronization error in the network was demonstrated.

6 Conclusion & Future Work

6.1 Thesis Summary

Internet-of-Things devices are being used in a growing number of applications due to their low cost and plug-and-play nature. Ensuring the reliability of a network of such devices is crucial for applications in critical services (*e.g.* fire sensor in a building has to reliably detect heat). To that end, this thesis focused on designing communication algorithms that dynamically adapt the network topology to minimize energy consumption and improve fault handling in such networks. Existing approaches were investigated and found to fall short in integrating all the major requirements discussed in Chapter 1. Without any prior knowledge of the network structure, each node exploits its position in the network to assume versatile roles such that the overall energy consumption and impact of faults in the network are minimized. To successfully develop communication solutions with all the three requirements, *i.e.*, energy efficiency, adaptability and fault resilience, the following major contributions have been presented in this thesis:

Clustering strategy: DeCoRIC. Given a completely decentralized ad-hoc IoT network, establishing communication with no synchronization among nodes results in very high power consumption and network interference as the devices keep their radios on continuously to send and receive messages. Additionally, the unstable nature of low-cost and resource-constrained devices leads to device failures that disrupt the connectivity among nodes and in turn affect the network functionality. Existing solutions to improve power efficiency include clustering and synchronization [23, 24, 22, 15]. However, these solutions only provide disparate properties such as adaptability and fault tolerance without an inclusive design of all the properties together.

We focused on clustering to establish a network structure that can aid in synchronization and improve fault tolerance. Existing clustering mechanisms used various parameters such as degree [44], residual energy [25], randomization [23], etc. to achieve a clustered architecture among the nodes. However, to maintain connectivity and ensure reliable routing for critical applications, the position of a node in the network is very important. We proposed DeCoRIC, a clustering mechanism that uses the degree of a node to identify nodes enabling connectivity and strives to cluster the neighboring nodes together such as that the end-to-end connectivity is maintained in a power-efficient process. DeCoRIC assumes an asynchronous ad-hoc network and utilizes the radio properties such as CSMA-CA and RDC to minimize the radio on-time while ensuring reliable message transmission. Furthermore, each node *gossips* its neighbor information to allow other nodes in the neighborhood to know the status of the nodes. The gossiping process ensures that the node failures are detected reliably within a bounded time and

transmitted to other nodes in the neighborhood to allow for re-routing/re-clustering. Additionally, any node's missed transmission is accounted for in this process as the radio on times of all the nodes may not be aligned. We compared the performance of our proposed solution with the state-of-the-art protocols LEACH and BEEM. DeCoRIC achieves up to 70% better power efficiency and 42% longer lifetime compared to LEACH while achieving up to 110% better power efficiency and 109% longer network lifetime in comparison to BEEM, respectively. To the best of our knowledge, DeCoRIC is the only clustering scheme that combines the properties of connectivity and resilience in an energy-efficient manner.

DeCoRIC use-case: Energy optimization with device mobility. Clustered architectures were typically tested on wireless sensor networks with limited applications on practical problems. We deployed an application with distributed energy resources and electric vehicles atop the clustered architecture generated using some features of DeCoRIC. While most works in the literature [79, 20, 98] focus on load shifting at the grid-level to minimize peak load and achieve load balancing, the device-level parameters have not been explored well. Due to load unpredictability and high skew, scheduling all devices at one location or time results in exceeding the peak load and blackouts. The flexibility of letting the devices decide the amount and location of power consumption based on the information from the grid allows significant improvement of benefit/utility for the devices.

In Chapter 4, we modeled a system of grid supply in the form of aggregators and devices with properties including different power consumption modes and mobility. An optimization problem was formulated with an objective to minimize the loss of utility incurred due to deadline misses and mobility, *i.e.,* additional power consumption due to movement between different aggregators. Synthetic data was generated to cover an exhaustive range of values for the input parameters. A low-complexity heuristic was proposed to solve the MINLP which can produce a feasible schedule within a practically viable runtime. The model was tested with a solver for a duration equivalent to the worst-case runtime of the heuristic and found that the utility loss was significantly high for the solver due to the large number of variables to optimize. A real-world EV dataset was used to test our proposed heuristic against standard scheduling algorithms and found that our solution achieved over 57.23% lower utility loss. Based on the existing literature, our solution is the first to optimize energy by utilizing device mobility and multiple modes of devices.

C-sync. Beyond the energy efficiency achieved by DeCoRIC, introducing time synchronization would further improve energy efficiency through the use of slotted communication, TDMA and extended sleep cycles. Most of the available synchronization solutions focus on accuracy and energy efficiency [22, 15, 23] but do not take into account the impact of faults in the network. To address the resilience and integrate the requisite properties discussed in Chapter 1, a synchronized clustered architecture with minimum power consumption and resilience against various faults in Section 2.3.2 was designed.

Building on DeCoRIC, we proposed C-sync, a resilient and energy-efficient synchronization solution for IoT devices in Chapter 5. The synchronization was done in two phases: clustering to establish the clustered architecture and the roles of every node and consensus phase to maintain synchronization and detect any faults in the network. A fault introduced in a CH node was detected successfully using the byzantine consensus mechanism. Theoretically, we proved the detection of faults in all scenarios as long as the assumptions were met. The performance of C-sync was compared with the state-of-the-art decentralized synchronization protocol GTSP. It was found that C-sync consumes at least 51% lower power than GTSP while having the same accuracy. Furthermore, using the concept of local centers, nodes at the center of the network were chosen to distribute time information such that the distance of any node to the time source is bounded. Through experiments, we showed that the measured synchronization error is lower than the worst-case estimated synchronization error due to multi-hop communication. To the best of our knowledge, C-sync is the only time synchronization solution that provides resilience against byzantine faults and has a configurable mechanism to limit hop count for a device from its time source.

To conclude, this thesis identified and described the problem of energy efficiency, adaptability, and fault resilience in networks of IoT devices. With growing adoption of resource-constrained devices in IoT applications, the three requirements are imperative for the stability and robustness of the network. Through solutions proposed in Chapters 3, 4 and 5, for the first time, we addressed these research gaps by developing a clustering scheme validated on both a wireless sensor network and an energy distribution network to be energy-efficient, adaptable, and resilient to faults concurrently.

Our proposed solution has wide-ranging implications for various IoT applications. For example, environmental sensing and industrial plants experience frequent failure of devices due to the harsh conditions in which the devices are deployed. Our resilient solution can adapt to these failures by switching to other devices in an energy-efficient way at scale.

6.2 Future work

While this thesis is the first to collectively combine all three major requirements of energy efficiency, adaptability and resilience, there is further potential to explore relevant research problems for the scientific community. In this section, we discuss some of the potential research topics that stem from this thesis.

6.2.1 DeCoRIC:

Multi-parameter clustering. In chapter 3, we proposed DeCoRIC which grouped nodes into clusters and elected cluster heads based on the degree of a node. The degree parameter ensured connectivity between the clusters and resilience in the clustered network. However, there are various other parameters such as residual energy of the nodes or position of a node within the network that can be used for clustering. Each of these parameters creates a different clustered network topology optimized for certain require-

ments such as energy efficiency, throughput, etc. As future work of DeCoRIC, multiple parameters for clustering can be combined and optimized to further improve the energy efficiency, adaptability and resilience of the network.

Alternate radio optimizations for energy-efficient communication. As we have seen in Chapter 3 and Chapter 5, clustering operation consumes the bulk of energy during communication between asynchronous devices. Radio duty cycling (RDC) provided a means to turn on/off the radio with different periods to conserve energy while ensuring successful transmission. However, RDC follows a fixed schedule for radio without considering communication patterns from neighboring devices. RDC can be further optimized to exploit the neighbor communication patterns and increase/decrease the on-period of the radio. The inclusion of a transmission window and/or enhancing the CSMA-CA with additional constraints reduces the interferences on the channel and minimizes the neighbor node detection time. Further, other radio-based optimizations such as transmission power and range, among others can be used to improve the energy efficiency of the devices.

6.2.2 Use-case of DeCoRIC - Load balancing for mobile devices:

Applications. With the exception of generic sensor networks which include homogeneous nodes, most applications comprise nodes with heterogeneous capabilities. For example, environment monitoring uses nodes not only with different sensing capabilities but also different connectivity (wireless/wired connectivity) and power sources (battery/electric line connections). The modular nature of the implementation can be used to expand the future applicability of the clustered architecture to accommodate such heterogeneous platforms.

Although the solution obtained from the proposed heuristic in Chapter 4 is fast and feasible, the quality of the solution is far from optimal. Standard optimization solvers produce optimal solutions at the cost of significant runtime. Hence, as future work, partial integration of the heuristic with the solver can speed up the solver and improve the utility loss closer to the optimal solution. To elaborate, the heuristic can generate an initial feasible solution and feed it to the solver for further optimization. This process makes away with the significant runtime needed for steps such as pre-solve.

6.2.3 C-sync:

Collision recovery. We observed through experiments in the previous chapters that interference between devices causes collisions and loss of messages. Although synchronization and slotted communication alleviate interference significantly, there is a limitation on the number of devices that can be supported with time slots. By utilizing the collided packets based on existing solutions on collision recovery [142], information can be extracted and recovered to minimize re-transmissions. Any reduction in transmission translates directly into a reduction in energy consumption of the nodes.

Enhanced resilience. As seen in Chapter 5, C-sync was able to detect a class of byzantine faults mostly observed in clustered architectures. However, the process of byzantine consensus is robust to achieve a more comprehensive fault tolerance beyond the type of faults discussed in Chapter 2. Additionally, improvements to the authentication mechanism of every node type (*e.g.* AES encryption applied to all messages) enhance the reliability of information transmitted from every node. The above changes coupled with improvements to the byzantine consensus expand the fault tolerance to the standard Byzantine Fault Tolerance (BFT) used in distributed systems. BFT not only detects and corrects faults but also enables the network to be resilient against malicious actors with targeted attacks.

Ground truth validation. A natural extension to improve the robustness of our solutions is the verification of ground truth. Fault resilience coupled with data verification can improve the detection of anomalous data by faulty/compromised IoT devices. Since we use a broadcast medium to transmit information across neighboring nodes, properties of the channel and the radio protocols (IEEE 802.15.4, 802.11, etc.) can be exploited to validate the data. Before utilizing the transmitted data, every node in the neighborhood can use the existing consensus to verify if the same changes to the radio and channel properties were observed. Furthermore, every node is equipped with a few sensors that can provide a context of the environment. The addition of sensing data with the existing broadcast medium can enhance the available modalities to improve the validation. Thus, anomalies observed in the broadcast data can be verified with any anomalies in the non-radio data from the sensing elements of the node.

As a long-term research direction, incorporating machine learning techniques help to understand the impact of various network parameters such as network size, communication frequency, etc., on the requirements of the IoT network. A comprehensively designed model can establish a self-organizing network by utilizing the structure of the network and communication primitives discussed in Chapter 2. Additionally, securing resource-constrained devices against malicious attacks with cryptographic hashing and multi-modal sensing can replace complex encryption techniques while preserving energy efficiency.

Bibliography

[1] International Telecommunications Union. ITU-T Y.4000/Y.2060, 2012. URL: http://handle.itu.int/11.1002/1000/11559.

[2] Ieee draft standard for ethernet amendment: Media access control parameters, physical layers and management parameters for 2.5 gb/s and 5 gb/s operation. *IEEE P802.3bz/D3.1, June 2016*, pages 1–198, 2016.

[3] Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low rate wireless personal area networks (wpans). *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pages 1–320, 2006. doi:10.1109/IEEESTD.2006.232110.

[4] Moteiv Corporation. Tmote sky. https://insense.cs.st-andrews.ac.uk/files/2013/04/tmote-sky-datasheet.pdf.

[5] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 138–149. ACM, 2003. doi:10.1145/958491.958508.

[6] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009. doi:https://doi.org/10.1016/j.adhoc.2008.06.003.

[7] X. Zhu, H. Han, S. Gao, Q. Shi, H. Cui, and G. Zu. A multi-stage optimization approach for active distribution network scheduling considering coordinated electrical vehicle charging strategy. *IEEE Access*, 6:50117–50130, 2018.

[8] F. K. W. Chan and H. C. So. Efficient weighted multidimensional scaling for wireless sensor network localization. *IEEE Transactions on Signal Processing*, 57(11):4548–4553, 2009. doi:10.1109/TSP.2009.2024869.

[9] H. Karvonen, J. Suhonen, J. Petäjäjärvi, M. Hämäläinen, M. Hännikäinen, and A. Pouttu. Hierarchical architecture for multi-technology wireless sensor networks for critical infrastructure protection. *Wireless Personal Communications*, 76:209–229, 2014. doi:10.1007/s11277-014-1686-2.

[10] A. Rullo, E. Serra, and J. Lobo. Redundancy as a measure of fault-tolerance for the internet of things: A review. *Policy-Based Autonomic Data Governance*, pages 202–226, 2019.

[11] Z. Chen, D. Zhang, R. Zhu, Y. Ma, P. Yin, and F. Xie. A review of automated formal verification of ad hoc routing protocols for wireless sensor networks. *Sensor Letters*, 11(5):752–764, 2013.

[12] J. Barnickel and U. Meyer. Secsywise: A secure time synchronization scheme in wireless sensor networks. In *2009 International Conference on Ultra Modern Telecommunications Workshops*, pages 1–8. doi:10.1109/ICUMT.2009.5345607.

[13] S. Bandyopadhyay and E. J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *Proc. IEEE Joint Conf. on Computer and Communications (INFOCOMM)*, volume 3, pages 1713–1723 vol.3, Mar. 2003.

[14] C. Lenzen, P. Sommer, and R. Wattenhofer. Pulsesync: An efficient and scalable clock synchronization protocol. *IEEE/ACM Transactions on Networking*, 23(3):717–727, 2015. doi:10.1109/TNET.2014.2309805.

[15] P. Sommer and R. Wattenhofer. Gradient clock synchronization in wireless sensor networks. In *2009 International Conference on Information Processing in Sensor Networks*, pages 37–48, 2009.

[16] Y. Li, G. Kumar, H. Hariharan, H. Wassel, P. Hochschild, D. Platt, S. Sabato, M. Yu, N. Dukkipati, P. Chandra, and A. Vahdat. Sundial: Fault-tolerant clock synchronization for datacenters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1171–1186, 2020.

[17] X. P. Gu and H. Zhong. Optimisation of network reconfiguration based on a two-layer unit-restarting framework for power system restoration. *Generation, Transmission and Distribution, IET*, 6:693–700, 07 2012. doi:10.1049/iet-gtd.2011.0591.

[18] X. Hu, T. Park, and K. G. Shin. Attack-tolerant time-synchronization in wireless sensor networks. In *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, pages 41–45. doi:10.1109/INFOCOM.2008.17.

[19] L. K. Alazzawi, A. M. Elkateeb, A. Ramesh, and W. Aljuhar. Scalability analysis for wireless sensor networks routing protocols. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*, pages 139–144, 2008.

[20] C. O. Adika and L. Wang. Smart charging and appliance scheduling approaches to demand side management. *International Journal of Electrical Power and Energy Systems*, 57:232 – 240, 2014. doi:https://doi.org/10.1016/j.ijepes.2013.12.004.

[21] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 39–49. ACM, 2004. doi:10.1145/1031495.1031501.

[22] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 73–84, 2011.

[23] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10 pp. vol.2–, 2000. doi:10.1109/HICSS.2000.926982.

[24] L. Xu, G. M. P. O'Hare, and R. Collier. A balanced energy-efficient multihop clustering scheme for wireless sensor networks. In *7th IFIP Wireless and Mobile Networking Conference*, pages 1–8, May 2014. doi:10.1109/WMNC.2014.6878886.

[25] O. Younis and S. Fahmy. HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks. *IEEE Trans. on Mobile Computing*, 3(4):366–379, Oct. 2004.

[26] J. He, J. Chen, P. Cheng, and X. Cao. Secure time synchronization in wireless sensor networks: A maximum consensus-based approach. *IEEE Transactions on Parallel and Distributed Systems*, pages 1055–1065, 2014. doi:10.1109/TPDS.2013.150.

[27] K. Fan, S. Wang, Y. Ren, K. Yang, Z. Yan, H. Li, and Y. Yang. Blockchain-based secure time protection scheme in iot. *IEEE Internet of Things Journal*, pages 4671–4679, 2019. doi:10.1109/JIOT.2018.2874222.

[28] S.Gholami, S. Saha, and M. Aldeen. Sensor fault tolerant control of microgrid. In *2016 IEEE Power and Energy Society General Meeting (PESGM)*, pages 1–5, 2016. doi:10.1109/PESGM.2016.7741809.

[29] R. van Renesse, Y. Minsky, and M. Hayden. A Gossip-Style Failure Detection Service. In *Proc. Middleware*, 1998.

[30] Z. J. Lee, T. Li, and S. Low. Acn-data: Analysis and applications of an open ev charging dataset. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, page 139–149, 2019. doi:10.1145/3307772.3328313.

[31] P. Appavoo, E. K. William, M. C. Chan, and M. Mohammad. Indriya2: A heterogeneous wireless sensor network (wsn) testbed. In *Testbeds and Research Infrastructures for the Development of Networks and Communities*, pages 3–19. Springer International Publishing, 2019.

[32] L. Xu, R. Collier, and G. M. P. O'Hare. A survey of clustering techniques in wsns and consideration of the challenges of applying such to 5g iot scenarios. *IEEE Internet of Things Journal*, pages 1229–1249, Oct. 2017. doi:10.1109/JIOT.2017.2726014.

[33] G. Bedi, G. K. Venayagamoorthy, R. Singh, R. R. Brooks, and K. Wang. Review of internet of things (iot) in electric power and energy systems. *IEEE Internet of Things Journal*, 5(2):847–870, 2018.

[34] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Trans. on Wireless Communications*, Oct. 2002. doi:10.1109/TWC.2002.804190.

[35] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava. Sensor network data fault types. *ACM Transactions on Sensor Networks*, 5(3), 2009. doi:10.1145/1525856.1525863.

[36] A. Mahapatro and P. M. Khilar. Fault diagnosis in wireless sensor networks: A survey. *IEEE Communications Surveys Tutorials*, 15(4):2000–2026, 2013.

[37] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. *Ad Hoc Networks*, 1(2):293 – 315, 2003. Sensor Network Protocols and Applications.

[38] L. Lamport, R. Shostak, and M.Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, jul 1982. doi:10.1145/357172.357176.

[39] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proc. of the IEEE Int. Conf. on Local Computer Networks*, pages 455–462, 2004.

[40] P. Suarez, C. Renmarker, A. Dunkels, and T. Voigt. Increasing zigbee network lifetime with X-MAC. In *Proceedings of the workshop on Real-world wireless sensor networks, RWSN '08, Glasgow, Scotland, April 1, 2008*, pages 26–30, 2008. doi:10.1145/1435473.1435481.

[41] Powertrace: Network-level Power Profiling for Low-power Wireless Networks. https://core.ac.uk/download/pdf/11435067.pdf.

[42] Zolertia corporation. http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf.

[43] E. Hartuv and R. Shamir. A Clustering Algorithm Based on Graph Connectivity. *Information processing letters*, 76(4-6), 2000.

[44] C.-Y. Wen and W. A. Sethares. Automatic Decentralized Clustering for Wireless Sensor Networks. *EURASIP Journal on Wireless Communications and Networking*, 2005(5), Dec. 2005.

[45] O. M. Alia, Z. Shaaban, A. Basheer, A. Al-Ajouri, and A. Alsswey. Musicians'-inspired clustering protocol for efficient energy Wireless Sensor Networks. In *Proc. Int. Conf. on Communications and Networking (ComNet)*, pages 1–6, Mar. 2014.

[46] T. M. Behera, S. K. Mohapatra, U. C. Samal, M. S. Khan, M. Daneshmand, and A. H. Gandomi. Residual energy-based cluster-head selection in wsns for iot application. *IEEE Internet of Things Journal*, 6(3):5132–5139, June 2019. doi:10.1109/JIOT.2019.2897119.

[47] A. A. A. Ari, B. O. Yenke, N. Labraoui, I. Damakoa, and A. Gueroui. A power efficient cluster-based routing algorithm for wireless sensor networks: Honeybees swarm intelligence based approach. *Journal of Network and Computer Applications*, 69:77–97, 2016.

[48] S. Lindsey, C. Raghavendra, and K. M. Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Transactions on Parallel and Distributed Systems*, 13:924–935, Sept. 2002. doi:10.1109/TPDS.2002.1036066.

[49] F. Avril, T. Bernard, A. Bui, and D. Sohier. Clustering and communications scheduling in WSNs using mixed integer linear programming. *Journal of Communications and Networks*, 16:421–429, 2014.

[50] R. S. Elhabyan and M. C. Yagoub. PSO-HC: Particle swarm optimization protocol for hierarchical clustering in wireless sensor networks. In *Proc. Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 417–424, 2014.

[51] D. Karaboga, S. Okdem, and C. Ozturk. Cluster based wireless sensor network routing using artificial bee colony algorithm. *Wireless Networks*, 18(7):847–860, 2012.

[52] X. Cai, Y. Duan, Y. He, J. Yang, and C. Li. Bee-sensor-c: An energy-efficient and scalable multipath routing protocol for wireless sensor networks. *Int. Journal of Distributed Sensor Networks*, page 976127, 2015. doi:10.1155/2015/976127.

[53] S. Yi, J. Heo, Y. Cho, and J. Hong. Peach: Power-efficient and adaptive clustering hierarchy protocol for wireless sensor networks. *Computer Communications*, 30(14):2842 – 2852, 2007. doi:https://doi.org/10.1016/j.comcom.2007.05.034.

[54] C. Li, M. Ye, G. Chen, and J. Wu. An energy-efficient unequal clustering mechanism for wireless sensor networks. In *IEEE Int. Conf. on Mobile Adhoc and Sensor Systems Conference, 2005.*, pages 8 pp.–604, Nov. 2005.

[55] M. A. Mirza and R. M. Garimella. PASCAL: Power Aware Sectoring Based Clustering Algorithm for Wireless Sensor Networks. In *Proc. Int. Conf. on Information Networking (ICOIN)*, pages 240–245, 2009.

[56] V. Loscri, G. Morabito, and S. Marano. A two-levels hierarchy for low-energy adaptive clustering hierarchy (TL-LEACH). In *Proc. IEEE Vehicular Technology Conf. (VTC)*, volume 3, pages 1809–1813, 2005.

[57] S. Al-Sodairi and R. Ouni. Reliable and energy-efficient multi-hop leach-based clustering protocol for wireless sensor networks. *Sustainable Computing: Informatics and Systems*, 20:1 – 13, 2018. doi:https://doi.org/10.1016/j.suscom.2018.08.007.

[58] M. Youssef, A. Youssef, and M. Younis. Overlapping Multihop Clustering for Wireless Sensor Networks. *IEEE Trans. on Parallel and Distributed Systems*, 20(12):1844–1856, Dec. 2009. doi:10.1109/TPDS.2009.32.

[59] H. Chan and A. Perrig. ACE: An Emergent Algorithm for Highly Uniform Cluster Formation. In *Proc. Wireless Sensor Networks*, 2004.

[60] S. Soro and W. B. Heinzelman. Prolonging the lifetime of wireless sensor networks via unequal clustering. In *19th IEEE Int. Parallel and Distributed Processing Symposium*, Apr. 2005. doi:10.1109/IPDPS.2005.365.

[61] G. Li and T. Znati. RECA: A Ring-Structured Energy-Efficient Cluster Architecture for Wireless Sensor Networks. In *Proc. Mobile Ad-hoc and Sensor Networks*, 2005.

[62] R. K. Jallu, P. R. Prasad, and G. K. Das. Distributed construction of connected dominating set in unit disk graphs. *Journal of Parallel and Distributed Computing*, 104, 2017. doi:https://doi.org/10.1016/j.jpdc.2017.01.023.

[63] X. Gao, X. Zhu, J. Li, F. Wu, G. Chen, D. Du, and S. Tang. A novel approximation for multi-hop connected clustering problem in wireless networks. *IEEE/ACM Tran. on Networking*, 25, Aug 2017. doi:10.1109/TNET.2017.2690359.

[64] T. Shi, S. Cheng, Z. Cai, and J. Li. Adaptive connected dominating set discovering algorithm in energy-harvest sensor networks. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016. doi:10.1109/INFOCOM.2016.7524504.

[65] L. Xu, M. J. O'Grady, G. M. P. O'Hare, and R. Collier. Reliable multihop intra-cluster communication for wireless sensor networks. In *2014 Int. Conf. on Computing, Networking and Communications (ICNC)*, pages 858–863, Feb. 2014. doi:10.1109/ICCNC.2014.6785450.

[66] A. N. Alvi, S. H. Bouk, S. H. Ahmed, and M. A. Yaqub. Influence of Backoff Period in Slotted CSMA/CA of IEEE 802.15.4. In *Int. Conf. on Wired/Wireless Internet Communication*, pages 40–51, May. 2016.

[67] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[68] H.-W. Tseng, Y.-C. Fan, S.-T. Sheu, and S.-Y. Ou. An effective grouping scheme for avoiding hidden node problem in ieee 802.15.4-based wireless sensor networks. *SIGAPP Appl. Comput. Rev.*, 14:30–40, Mar. 2014. doi:10.1145/2600617.2600620.

[69] Networking the iot with ieee 802.15.4/6lowpan. Technical report.

[70] A. Riker, M. Curado, and E. Monteiro. Neutral operation of the minimum energy node in energy-harvesting environments. In *IEEE Symposium on Computers and Communication (ISCC)*, July 2017.

[71] J. Shin, U. Ramachandran, and M. Ammar. On improving the reliability of packet delivery in dense wireless sensor networks. In *16th Int. Conf. on Computer Communications and Networks*, pages 718–723, Aug. 2007. doi:10.1109/ICCCN.2007.4317902.

[72] I. S. Bayram, G. Michailidis, and M. Devetsikiotis. Unsplittable load balancing in a network of charging stations under qos guarantees. *IEEE Transactions on Smart Grid*, 6(3):1292–1302, 2015.

[73] IEA. Global energy review 2020, 2020. URL: https://www.iea.org/reports/global-energy-review-2020.

[74] S. Rafique, M. S. H. Nizami, U. B. Irshad, M. J. Hossain, and S. C. Mukhopadhyay. Ev scheduling framework for peak demand management in lv residential networks. *IEEE Systems Journal*, pages 1–9, 2021. doi:10.1109/JSYST.2021.3068004.

[75] W. E. P. COMPANY, 2020. URL: https://www.we-energies.com/pdfs/etariffs/wisconsin/elecrateswi.pdf.

[76] Y. Liu, C. Yuen, N. Ul Hassan, S. Huang, R. Yu, and S. Xie. Electricity cost minimization for a microgrid with distributed energy resource under different information availability. *IEEE Transactions on Industrial Electronics*, 62(4):2571–2583, 2015.

[77] R. Deng, Z. Yang, M. Chow, and J. Chen. A survey on demand response in smart grids: Mathematical models and approaches. *IEEE Transactions on Industrial Informatics*, 11:570–582, 2015.

[78] F. Ye, Y. Qian, and R. Q. Hu. A real-time information based demand-side management system in smart grid. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):329–339, 2016.

[79] M. López, S. de la Torre, S. Martín, and J. Aguado. Demand-side management in smart grid operation considering electric vehicles load shifting and vehicle-to-grid support. *International Journal of Electrical Power and Energy Systems*, 64:689 – 698, 2015. doi:https://doi.org/10.1016/j.ijepes.2014.07.065.

[80] Y. Zhang and L. Cai. Dynamic charging scheduling for ev parking lots with photovoltaic power system. *IEEE Access*, 6, 2018.

[81] E. AB. URL: https://www.winningappliances.com.au/public/manuals/Electrolux-ETM4200SDRH-420L-Fridge-User-Manual.pdf.

[82] W. Gay. *Raspberry Pi Hardware Reference*. 01 2014. doi:10.1007/978-1-4842-0799-4.

[83] Espressif Systems. *ESP32 Thing*, 10 2016. Ver. 1.0.

[84] Texas Instruments. *INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I2C Interface*, 12 2015.

[85] M. Yu and S. H. Hong. Supply–demand balancing for power management in smart grid: A stackelberg game approach. *Applied Energy*, 164:702 – 710, 2016. doi:https://doi.org/10.1016/j.apenergy.2015.12.039.

[86] S. Noor, W. Yang, M. Guo, K. H. van Dam, and X. Wang. Energy demand side management within micro-grid networks enhanced by blockchain. *Applied Energy*, 228:1385 – 1398, 2018. doi:https://doi.org/10.1016/j.apenergy.2018.07.012.

[87] D. Zhao, H. Wang, J. Huang, and X. Lin. Pricing-based energy storage sharing and virtual capacity allocation. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, 2017.

[88] J. Kim and Y. Dvorkin. Enhancing distribution resilience with mobile energy storage: A progressive hedging approach. In *2018 IEEE Power Energy Society General Meeting (PESGM)*, pages 1–5, 2018.

[89] C. Li, X. Yu, W. Yu, G. Chen, and J. Wang. Efficient computation for sparse load shifting in demand side management. *IEEE Transactions on Smart Grid*, 8(1):250–261, 2017.

[90] H. K. Nguyen, J. B. Song, and Z. Han. Distributed demand side management with energy storage in smart grid. *IEEE Transactions on Parallel and Distributed Systems*, 26(12):3346–3357, 2015.

[91] H. Roh and J. Lee. Residential demand response scheduling with multiclass appliances in the smart grid. *IEEE Transactions on Smart Grid*, 7(1):94–104, 2016.

[92] P. He, M. Li, L. Zhao, B. Venkatesh, and H. Li. Water-filling exact solutions for load balancing of smart power grid systems. *IEEE Transactions on Smart Grid*, 9(2):1397–1407, 2018.

[93] T. Chiu, Y. Shih, A. Pang, and C. Pai. Optimized day-ahead pricing with renewable energy demand-side management for smart grids. *IEEE Internet of Things Journal*, 4(2):374–383, 2017.

[94] A. Ogunjuyigbe, T. Ayodele, and O. Akinola. User satisfaction-induced demand side load management in residential buildings with user budget constraint. *Applied Energy*, 187:352 – 366, 2017. doi:https://doi.org/10.1016/j.apenergy.2016.11.071.

[95] H. A. Özkan. Appliance based control for home power management systems. *Energy*, 114:693 – 707, 2016.

[96] M. H. K. Tushar, A. W. Zeineddine, and C. Assi. Demand-side management by regulating charging and discharging of the ev, ess, and utilizing renewable energy. *IEEE Transactions on Industrial Informatics*, 14(1):117–126, 2018.

[97] A. Schuller, C. M. Flath, and S. Gottwalt. Quantifying load flexibility of electric vehicles for renewable energy integration. *Applied Energy*, 151:335 – 344, 2015. doi:https://doi.org/10.1016/j.apenergy.2015.04.004.

[98] Y. Zheng, Y. Song, D. J. Hill, and K. Meng. Online distributed mpc-based optimal scheduling for ev charging stations in distribution systems. *IEEE Transactions on Industrial Informatics*, 15(2):638–649, 2019.

[99] B. Alinia, M. H. Hajiesmaili, and N. Crespi. Online ev charging scheduling with on-arrival commitment. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4524–4537, 2019.

[100] L. Yao, W. H. Lim, and T. S. Tsai. A real-time charging scheme for demand response in electric vehicle parking station. *IEEE Transactions on Smart Grid*, 8(1):52–62, 2017.

[101] F. Rassaei, W. Soh, and K. Chua. Demand response for residential electric vehicles with random usage patterns in smart grids. *IEEE Transactions on Sustainable Energy*, 6(4):1367–1376, 2015.

[102] R. Deng and H. Liang. Whether to charge or discharge an electric vehicle? an optimal approach in polynomial time. In *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, pages 1–5, 2017.

[103] H. Yi, Q. Lin, and M. Chen. Balancing cost and dissatisfaction in online ev charging under real-time pricing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1801–1809, 2019.

[104] H. M. Salkin and C. A. De Kluyver. The knapsack problem: a survey. *Naval Research Logistics Quarterly*, 22(1):127–144, 1975.

[105] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24777-7.

[106] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL: https://www.gurobi.com.

[107] G. Van Rossum and F. L. Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[108] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems Journal*, 30:129–154, 2005.

[109] D. Griffin, I. Bate, and R. I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 76–88, 2020. doi:10.1109/RTSS49844.2020.00018.

[110] S. E. M. Authority, 2019. URL: https://www.ema.gov.sg/cmsmedia/Publications_and_Statistics/Statistics/8RSU.pdf.

[111] E. V. Database, 2021. URL: https://ev-database.org/.

[112] B. Moussa, M. Debbabi, and C. Assi. Security assessment of time synchronization mechanisms for the smart grid. *IEEE Communications Surveys & Tutorials*, pages 1952–1973, 2016. doi:10.1109/COMST.2016.2525014.

[113] Z. Wang, P. Zeng, M. Zhou, and D. Li. Cluster-based maximum consensus time synchronization in iwsns. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2016. doi:10.1109/VTCSpring.2016.7504162.

[114] J. Wu, L. Zhang, Y. Bai, and Y. Sun. Cluster-based consensus time synchronization for wireless sensor networks. *IEEE Sensors Journal*, 15(3):1404–1413, 2015. doi:10.1109/JSEN.2014.2363471.

[115] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, Oct 1991.

[116] L. Schenato and F. Fiorentin. Average timesynch: A consensus-based protocol for clock synchronization in wireless sensor networks. *Automatica*, 47(9):1878 – 1886, 2011.

[117] S. Ganeriwal, C. Pöpper, S. Čapkun, and M. B. Srivastava. Secure time synchronization in sensor networks. *ACM Transactions on Information and System Security*, 2008. doi:10.1145/1380564.1380571.

[118] Q. Li and D. Rus. Global clock synchronization in sensor networks. *IEEE Transactions on Computers*, pages 214–226, 2006. doi:10.1109/TC.2006.25.

[119] E. Regnath, N. Shivaraman, S. Shreejith, A. Easwaran, and S. Steinhorst. Blockchain, what time is it? trustless datetime synchronization for iot. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–6. doi:10.1109/COINS49042.2020.9191420.

[120] C. G. Ramirez, A. Dyussenova, A. Sergeyev, and B. Iannucci. Longshot: Long-range synchronization of time. In *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 289–300, 2019.

[121] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Operating Systems Review*, 36(SI):147–163, 2002. doi:10.1145/844128.844143.

[122] K. S. Yildirim and A. Kantarci. Time synchronization based on slow-flooding in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):244–253, 2014.

[123] P. Yadav, J. A. McCann, and T. Pereira. Self-synchronization in duty-cycled internet of things (iot) applications. *IEEE Internet of Things Journal*, 4(6):2058–2069, 2017.

[124] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, 2004. doi:10.1109/TAC.2004.834113.

[125] W. Ren and R. W. Beard. Consensus of information under dynamically changing interaction topologies. In *Proceedings of the 2004 American Control Conference*, volume 6, pages 4939–4944 vol.6, 2004.

[126] S. Dasarathan, M. Banavar, C. Tepedelenlioglu, and A. Spanias. Distributed average consensus using bounded transmissions. In *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 1202–1206, 2012. doi:10.1109/ACSSC.2012.6489212.

[127] L. Schenato and G. Gamba. A distributed consensus protocol for clock synchronization in wireless sensor network. In *2007 46th IEEE Conference on Decision and Control*, pages 2289–2294, 2007. doi:10.1109/CDC.2007.4434671.

[128] K. Xie, Q. Cai, and M. Fu. A fast clock synchronization algorithm for wireless sensor networks. *Automatica*, 92:133 – 142, 2018.

[129] W. Ren and R. W. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, 50(5):655–661, 2005. doi:10.1109/TAC.2005.846556.

[130] M. El Chamie, G. Neglia, and K. Avrachenkov. Reducing communication overhead for average consensus. In *2013 IFIP Networking Conference*, pages 1–9, 2013.

[131] J. H. Cho, H. Kim, S. Wang, J. Lee, H. Lee, S. Hwang, S. Cho, Y. Oh, and T. J. Lee. A novel method for providing precise time synchronization in a distributed control system using boundary clock. *IEEE Transactions on Instrumentation and Measurement*, 58(8):2824–2829, 2009.

[132] S. Viswanathan, R. Tan, and D. K. Y. Yau. Exploiting power grid for accurate and secure clock synchronization in industrial iot. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 146–156, 2016. doi:10.1109/RTSS.2016.023.

[133] F. Terraneo, F. Riccardi, and A. Leva. Jitter-compensated vht and its application to wsn clock synchronization. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 277–286, 2017. doi:10.1109/RTSS.2017.00033.

[134] M. Buevich, N. Rajagopal, and A. Rowe. Hardware assisted clock synchronization for real-time sensor networks. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 268–277, 2013. doi:10.1109/RTSS.2013.34.

[135] N. Shivaraman, S. Ramanathan, S. Shanker, A. Easwaran, and S. Steinhorst. Decoric: Decentralized connected resilient iot clustering. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–10, 2020.

[136] J. Yin, J. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault tolerant services. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 253–267, 2003.

[137] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation*, 118(1):158 – 179, 1995.

[138] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the IEEE International Conference on Local Computer Networks*, pages 455–462, 2004.

[139] Texas Instruments. *MSP430 Microcontroller*. URL: https://www.ti.com/product/MSP430F1611.

[140] Texas Instruments. *CC2420 RF Tranceiver*. URL: https://www.ti.com/product/CC2420.

[141] N. Panigrahi and P. M. Khilar. Multi-hop consensus time synchronization algorithm for sparse wireless sensor network: A distributed constraint-based dynamic programming approach. *Ad Hoc Networks*, 61:124 – 138, 2017. doi:https://doi.org/10.1016/j.adhoc.2017.04.002.

[142] S. Gollakota and D. Katabi. Zigzag decoding: Combating hidden terminals in wireless networks. *SIGCOMM Comput. Commun. Rev.*, 38(4), 2008. doi:10.1145/1402946.1402977.