Data Redundancy Management in Connected Environments

Faisal Shahzad

UNIV PAU & PAYS ADOUR

DOCTORAL THESIS

Data Redundancy Management in Connected Environments

Author: Faisal SHAHZAD

Advisors:	Pr. Richard Chbeir	University Pau & Pays Adour, France
	Dr. Joe Tekli	Lebanese American University, Lebanon
Collaborator:	Dr. Elio Mansour	University Pau & Pays Adour, France
Reviewers:	Pr. Djamal Benslimane	University of Lyon, France
	Dr. Ahmed Mostefaoui	University of Franche Comté, France
Examiner:	Pr. Allel HADJALI	Ecole Nat Sup Mecanique Aerotechnique, France

A thesis submitted in fulfillment of the requirements for the degree in Computer Science

January 10, 2022

To my lovely Family…

I dedicate this work to my father Tilawat Khan and my mother Farhat Tilawat Khan, to my brothers and sisters and to my lovely wife and kids

Acknowledgements

Before discussing the scientific details of this research, I would like to take this opportunity, to thank the people and institutions that were pivotal to this work. . .

I would like to start by thanking Pr. Djamal BENSLIMANE and Dr. Ahmed MOSTEFAOUI for the time and effort they dedicated to the examination of my dissertation. I also thank them for their comments, suggestions, and feedback. Moreover, I am grateful for the presence of Pr. Allel HADJALI and Dr. Joe TEKLI in my thesis evaluation committee. Your comments and input are highly appreciated.

I would like to pay my gratitude to my supervisors Pr. Richard CHBEIR, Dr. Joe Tekli and collaborator Dr. Elio Mansour. I want to thank Richard for having faith in me, helping me get most out of my abilities, and insisting on elevating my skills and overall approach towards research. His guidance, insightful comments, bright ideas, and immense knowledge made this work possible. Aside the scientific aspects, I also want to thank him for understanding me and discussing things other than research work, always encouraging me, to push myself and do good work. I want to sincerely thank Joe, who guided me whenever I had problems. His problem solving skills and ideas always helped me think differently about the problem. He constantly insisted, to stay motivated and dig deep inside the research. My doctoral journey would not have been possible and enjoyable without his support and guidance. It was a great exposure for me, working with both of them and I look forward to work with them in future. I also want to thank and appreciate the efforts of Elio. His involvement in the project gave it a whole new dimension. He always guided me in how to analyse things critically and think of broader vision when comparing things. I really got very helpful ideas from his discussions. He made things very simple and easy for me whenever I was stuck in solving the problems. I learnt a lot of things from him as well and wish to work with him in future as well.

I am thankful for being part of the LIUPPA laboratory. I owe my gratitude to the "E2S" the funding body of the project. Nothing would have been possible without its financial support and interest in promoting science. I deeply appreciate the help and support of Dr. Sebastien Laborie and Dr. Christian Sallabery, who always contributed good ideas, during the discussions of the project. I greatly benefited from their ideas as well. I would also like to thank Sylvana YAKHNI, who worked on Fuzzy rules in FREDD framework in the later part of the thesis.

I also want to thank the "IUT de Bayonne et du Pays Basque" in Anglet for allowing me to use the materials and facilities. I would like to name Philippe Aniorté, Patrick Etcheverry, Philippe Lopistéguy and my colleagues in the research open space. I acknowledge the moral and emotional support provided by Sabri Allani, Nabila Guenonni, Paulo, Umbertho, Lara Kallab, and Karam Bou Chaya. These people were at my side every time I needed them. I would also not like to forget my friends in Pakistan. I would like to thank specially, Khalid Iqbal, Farman Marwat, Qasim Khan and Muhammad Hafeez, who always encouraged me to do doctoral studies from abroad. Their support and motivation enabled me to achieve this goal.

I sincerely appreciate my friends here in France. Specially, many thanks goes to Saeed Ullah, who became a brother during these years. His cool minded nature

and patience have helped me a lot to get out of tough times. He always gave good suggestions and ideas, and helped me in every way to get out of bad patch, whenever there was one. Besides him, I also want to thank Rehman, Lutf Ur Rehman, Muhammad Iqbal, Ghulam Murtaza, Arshad Hussain, Ahsan Saif and Muhmmada Rizwan, for their support and laughter that they brought on my faces in tensed situation. Finally, I would also like to thank Martin WALTON, for his unconditional guidance and support in every matter.

I want to thank my parents, who did everything possible to make me a good and successful person. They supported me in every way and always encouraged me to do what I wanted to do. I can never repay them in my whole life. I am short of words, to describe their sacrifices, endurance and everything they gave up for me. I always pray to ALLAH almighty, to bless them...

And last but definitely not least, my better half, my lovely wife Ayesha SHAHZAD. I would have never been able to do, whatever I did in this doctoral journey. She always stood besides me and beard me in my bad times. I cannot repay you, for your efforts in these years and I always love you a lot. At last, to my beautiful daughters, Abrish SHAHZAD and Fajr SHAHZAD, whom I missed a lot in these years by being far from them. I just want to tell them, I love them very much and I am sorry for not being with you in these years. This journey was hard without you two, but it was to give you a very good and successful future. From the bottom of my heart, I thank you my lovely wife and my beautiful daughters...

Abstract

Major advances in the fields of Internet and Communication Technology (ICT), data modeling/processing, and sensing technology have rendered traditional environments (e.g., cities, buildings, hospitals) more connected. In such environments, the ever-increasing number and diversity of data sources (devices and sensors, both static and mobile) is continuously increasing the size of the data that need to be stored locally on edge devices and/or transmitted to central storage databases/repositories in the network. Moreover, sensors monitoring the environment in real-time often produce redundant data that might not be necessarily useful (e.g., recording the same unchanged phenomena during a period of time). As a result, the huge amount of data causes several issues (e.g., network bandwidth saturation, over-consumption of network/device energy, cloud storage and I/O throughput issues). Data pre-processing techniques such as redundancy detection and cleaning (i.e., deduplication) can help reduce the amount of data being processed, transmitted, and stored in connected environments.

In this thesis, we focus on three main challenges: (i) accurately detecting data redundancies in connected environments; (ii) considering the physical environmental features (e.g., zone/location separations), the sensor features (e.g., coverage areas), and the dynamicity of the connected environment (e.g., device mobility) when detecting redundancies; and (iii) providing flexible and configurable redundancy cleaning mechanisms to cope with data consumer needs (e.g., users, services, devices, storage needs).

In order to address the aforementioned challenges, we first target data redundancies at the edge of the network (i.e., device level) and propose a Data Redundancy Management Framework (DRMF) to detect crisp data redundancies (e.g. temporal in case of static devices and spatio-temporal in case of mobile devices). Then, we address the limitations of DRMF by proposing FREDD (Fuzzy Redundancy Elimination for Data Deduplication) to detect fuzzy data redundancies at the device level. Furthermore, we consider the physical features of the environment, as well as sensor mobility and coverage areas, in an extension of FREDD that aims at detecting redundancies at the sink level. Finally, we propose several redundancy cleaning mechanisms to cope with the needs of data consumers (e.g., users, devices, services, and databases).

We form a global framework called DRMCE that groups the aforementioned modules for data redundancy detection in connected environments. Our proposal can be used with different connected environments such as buildings, cities, and hospitals.

Résumé

L'évolution significative des technologies de traitement, transmission, et modélisation des données ainsi que l'avancement technique réalisé au niveau des réseaux de capteurs ont contribué à la prolifération des environnements connectés (ex : bâtiments, villes intelligents). Ces environnements contiennent divers capteurs qui surveillent notre vie quotidienne et génèrent une quantité massive de données / observations. Par conséquent, le nombre de données à transmettre et stocker sur le réseau (localement sur les capteurs ou dans des bases de données) devient problématique. De plus, les capteurs surveillant l'environnement en temps réel produisent souvent des données redondantes qui ne sont pas nécessairement utiles (par exemple, enregistrer les mêmes phénomènes inchangés pendant une période de temps). Ceci provoque plusieurs problèmes (par exemple, la saturation de la bande passante du réseau, la surconsommation d'énergie du réseau / capteurs, des problèmes de stockage). Les techniques de prétraitement des données telles que la détection et le nettoyage de la redondance (c'est-à-dire la déduplication) pourraient aider à réduire la quantité de données traitées, transmises et stockées dans des environnements connectés.

Dans cette thèse, nous nous concentrons sur trois défis principaux: (i) détecter avec précision les redondances de données dans des environnements connectés ; (ii) prendre en compte les caractéristiques environnementales physiques (par exemple, les séparations des zones/localisations), les caractéristiques des capteurs (par exemple, les zones de couverture), et la dynamique de l'environnement connecté (par exemple, la mobilité des capteurs) lors de la détection des redondances ; et (iii) fournir des mécanismes flexibles et configurables de nettoyage des redondances pour faire face aux besoins des consommateurs de données (par exemple, les utilisateurs, les services, les équipements, les besoins de stockage).

Pour ce faire, nous proposons :

- une approche, dénotée DRMF, pour la détection des données redondantes au niveau des capteurs,
- une amélioration de DRMF, dénotée FREDD, basée sur le raisonnement flou pour améliorer la détection au niveau des capteurs,
- une extension de FREDD permettant de détecter les redondances au niveau intermédiaire (niveau "Sink"), et
- une approche pour le nettoyage des redondances détectées qui assure une flexibilité pour adapter la déduplication aux besoins et recommandations des consommateurs de données.

Nous regroupons tous ces modules dans un framework global nommé DRMCE pour la gestion de la redondance des données dans des environnements connectés. Notre proposition est 1) adaptative car permet de personnaliser la déduplication des données redondantes soit localement sur les capteurs ou au niveau des nœuds Sinks, et 2) générique car pourrait être utilisée dans différents environnements connectés tels que les bâtiments, les villes, et parkings intelligents.

Le manuscrit est organisé comme suit:

Chapitre 1

Introduction

Dans ce chapitre, nous introduisons les environnements connectés basés sur l'Internet des Objets et les réseaux de capteurs. Nous évoquons les éléments qui constituent ces environnements afin de clarifier la terminologie utilisée dans le manuscrit. Par la suite, nous présentons quelques problèmes liés à la qualité de données pour se focaliser sur la redondance de données. Ce chapitre introduit la motivation de ce travail de recherche. Pour ce faire, nous adoptons un exemple d'environnement connecté (un hôpital intelligent dans lequel une diversité de capteurs, mobiles et immobiles, surveillent l'environnement et génèrent des données / observations) qui sera un fil conducteur tout le long du manuscrit pour illustrer la motivation derrière chaque chapitre. Nous y montrons un ensemble de besoins liés à la détection de la redondance au niveau de capteurs à l'extrémité du réseau et au niveau des équipements centraux au cœur du réseau. Ce scénario montre aussi les défis techniques et scientifiques que nous considérons afin d'adresser les besoins susmentionnés. Ensuite, nous discutons les spécificités physiques de l'environnement et des capteurs, et leurs impacts sur la détection précise des redondances. Après, nous adressons les besoins liés à la déduplication et le nettoyage flexible des redondances tout en considérant les besoins des consommateurs des données afin de leur livrer des ensembles de données non redondantes et compatibles avec leurs spécifications. Finalement, ce chapitre présente le framework DRMCE, récapitule les contributions, et détaille l'organisation du rapport.

Chapitre 2

État de l'art

Dans ce chapitre, nous évaluons les approches existantes dans la littérature qui se focalisent sur la gestion de la redondance des données dans des environnements connectés et dans d'autres environnements similaires. Nous catégorisons les approches pour comparer les méthodes existantes pour la détection des redondances au niveau de la source (à l'extrémité du réseau) et au niveau intermédiaire/central. De plus, nous consacrons une sous-section pour le nettoyage des redondances détectées (la déduplication). Afin de pouvoir comparer ces approches, nous définissons en ensemble de critères qui répondent aux besoins et défis mentionnés dans le scénario de motivation présenté dans le chapitre précédent.

Chapitre 3

DRMF

Dans ce chapitre, nous présentons DRMF: "Data Redundancy Management Framework". Cette approche détecte les redondances des données localées sur le capteur pour éviter les transmissions non nécessaires vers le réseau. De cette façon, la détection a lieu à la source (sur les capteurs immobiles et mobiles). Dans cette première contribution, nous évaluons la distance entre les valeurs des observations générées

par un capteur afin de créer des clusters de données redondantes. Les périmètres des clusters sont définis par des seuils ("thresholds") rigides pour permettre une détection simple des redondances. En revanche, la précision aux extrémités des clusters peut être améliorée pour éviter d'affecter une donnée non redondante à un cluster de redondances.

Chapitre 4

FREDD

Dans ce chapitre, nous introduisons FREDD: "Fuzzy Redundancy Elimination for Data Deduplication" afin d'adresser la limitation de DRMF. Nous utilisons la logique floue pour détecter les données redondantes de façon plus précise au niveau des capteurs. Comme dans DRMF, nous considérons dans FREDD la mobilité des capteurs et la diversité des données captées. Ensuite, nous comparons la précision de la détection des redondances avec DRMF et d'autres approches existantes.

Chapitre 5

FREDD – Cœur du réseau

Dans ce chapitre, nous considérons la structure physique de l'environnement connecté, les aspects géographiques (tel que la carte de géolocalisation), les séparateurs physiques et virtuels entre les différentes localisations de l'environnement, ainsi que les aspects techniques des capteurs (tel que les zones de couvertures). Ceci permet d'étendre FREDD afin de considérer la détection précise des redondances au niveau intermédiaire du réseau et ne se limite pas au niveau des capteurs (localisés à l'extrémité du réseau).

Chapitre 6

Nettoyage des redondances

Dans ce chapitre, nous discutons le nettoyage des redondances détectées préalablement en proposant un mode de déduplication automatique et un autre mode basé sur les besoins des consommateurs de données. Qu'ils s'agissent des utilisateurs qui requêtent l'environnement, des services qui traitent les données, des équipements qui échangent les données, ou bien des supports de stockage, chaque consommateur de données pourrait avoir des besoins spécifiques qui affectent la déduplication (par exemple, un service qui génère des statistiques doit prendre en compte les valeurs redondantes pour ne pas biaiser les moyennes. En revanche, une base de données pourra nécessiter une déduplication complète des données pour ne pas épuiser ses ressources).

Chapitre 7

Conclusion & travaux futurs

Ce chapitre résume les différentes contributions de la thèse et discute à la fois les extensions futures ainsi que les nouvelles orientations possibles pour la suite de ce travail de recherche.

Contents

Acknowledgements **ii**

Abstract **iv**

Résumé **v**

1 Introduction **1**
1.1 Connected Environments, IoT, & Sensor Networks 1
1.1.1 Key Elements in IoT 3
Things 3
Gateways 3
Network Infrastructures 4
Cloud Infrastructures 4
1.1.2 Sensor Networks 4
1.2 Overview of Data Inconsistencies 5
1.2.1 Missing Values 5
1.2.2 Anomalies 5
1.2.3 Data Redundancies 5
1.3 Thesis Context 6
1.3.1 Objectives 6
1.3.2 Motivation Scenario 6
1.4 Proposal 8
1.4.1 Redundancy Detection at the Edge 8
1.4.2 Redundancy Detection at the Sink 9
1.4.3 Redundancy Cleaning 9
1.5 Report Organization 9

2 Related Works **11**
2.1 Deduplication in Data Storage and Data Warehousing 11
2.2 Redundancy Detection in Connected Environments 12
2.2.1 Redundancy Detection at the Network Edge 12
2.2.2 Redundancy Detection at the Network Sink 13
2.2.3 Redundancy Detection Comparison 16
2.3 Data Redundancy Cleaning 17
2.3.1 Data Redundancy Cleaning at the Network Edge 17
2.3.2 Data Redundancy Cleaning at the Network Sink 19
2.3.3 Data Redundancy Cleaning Comparison 20
2.4 Recap 20

3 DRMF: Data Redundancy Management Framework **22**
3.1 Introduction 23
3.2 Preliminaries 24
3.3 DRMF: Data Redundancy Management for leaF-edge 27

3.3.1 Data Type Filtering . . . 27
3.3.2 Redundancy Detection . . . 28
Redundancy Detection Algorithms . . . 28
3.3.3 Redundancy Detection Tuning . . . 31
3.4 DRMF Evaluation . . . 31
3.4.1 Complexity Evaluation . . . 31
Algorithm 1 Evaluation . . . 32
Algorithm 2 Evaluation . . . 32
Theoretical Complexity Comparison . . . 33
3.4.2 Experiments and Results . . . 33
Experimental Protocol . . . 34
Performance Evaluation . . . 34
Accuracy Evaluation . . . 36
3.5 Recap . . . 37

4 FREDD: Fuzzy Redundancy Elimination for Data Deduplication 38
4.1 Introduction . . . 39
4.2 Fuzzy Logic Preliminaries . . . 39
4.3 FREDD Framework . . . 40
4.3.1 Sensor Data Representation . . . 41
4.3.2 Attribute Separation . . . 41
4.3.3 Pattern Code Generation . . . 42
Value Pattern Codes . . . 42
Zone Pattern Codes . . . 42
Combined Pattern Codes . . . 43
4.3.4 Duplicate Candidate Filtering . . . 44
4.3.5 Fuzzy Redundancy Detection . . . 45
Fuzzy Inference Agent . . . 45
Computation Example . . . 48
4.4 FREDD Evaluation . . . 51
4.4.1 Complexity Evaluation . . . 51
4.4.2 Experimental Evaluation . . . 51
Experimental Test Data . . . 51
Evaluation Metrics . . . 52
Quality Evaluation . . . 53
Performance Evaluation . . . 55
4.5 Recap . . . 55

5 Detecting Data Redundancies at the Sink Level 57
5.1 Introduction . . . 58
5.2 Sink Level Use Cases . . . 59
5.2.1 Case 1: Zone-based with Hard Zone Separations . . . 60
5.2.2 Case 2: Zone-and-Coverage based with Hard Zone Separations 60
5.2.3 Case 3: Zone-based with Soft Zone Separations . . . 60
5.2.4 Case 4: Zone-and-Coverage based with Soft Zone Separations . 60
5.3 FREDD Sink Level Process . . . 61
5.4 FREDD Sink Level Evaluation . . . 62
5.4.1 Complexity Evaluation . . . 62
5.4.2 Experimental Evaluation . . . 62
Experimental Test Data . . . 62
Evaluation Metrics . . . 63

Quality Evaluation 63
Performance Evaluation 69
Baseline Comparison with Existing Approaches 70
5.5 Recap 72

6 Data Redundancy Cleaning **73**
6.1 Introduction 74
6.2 Preliminaries & Illustration Example 75
6.3 Linking Deduplication to DRMF & FREDD 77
6.3.1 Redundancy Cleaning Components 78
6.4 Implementation & Preliminary Evaluation 79
6.5 Recap 80

7 Conclusion & Future Works **81**
7.1 Report Recap 81
7.1.1 In Chapter 1 81
7.1.2 In Chapter 2 81
7.1.3 In Chapter 3 81
7.1.4 In Chapter 4 82
7.1.5 In Chapter 5 82
7.1.6 In Chapter 6 82
7.2 Future Research Directions 82
7.2.1 DRMF 82
7.2.2 FREDD 83
7.2.3 Redundancy Cleaning 83
7.2.4 Other Research Directions 83

A FREDD Computation Example **84**
A.1 Computation Example - Temperature Collection 84
A.1.1 Fuzzification: 84
A.1.2 Condition-Action Rules: 84
A.1.3 Inference: 85
A.1.4 Aggregation: 86
A.1.5 Defuzzification: 86
A.1.6 Result: 86

Bibliography **88**

List of Figures

1.1 CE impact on various application domains . . . 2
1.2 IoT projected market share by 2025 . . . 3
1.3 Connected Environment Network Representation . . . 4
1.4 Smart Hospital Example . . . 7
1.5 DRMCE Framework . . . 8

3.1 Data Processing in Connected Environments . . . 23
3.2 Processing of DRMF . . . 27
3.3 Performance Results: Experiments 1 and 2 . . . 35
3.4 Experiment 3 Results . . . 36
3.5 Experiment 4 Results . . . 36
3.6 Experiment 5 Results . . . 37

4.1 FREDD's Architecture . . . 40
4.2 Smart Hospital Zone Divisions . . . 43
4.3 Fuzzy Redundancy Detection Overview . . . 45
4.4 Fuzzy Partitions Using The Trapezoidal Function . . . 46
4.5 Fuzzification of Humidity *DataItem*1 and *DataItem*2 . . . 48
4.6 Inference - Rule2 . . . 49
4.7 Inference - Rule3 . . . 50
4.8 Aggregation . . . 50
4.9 Defuzzification . . . 50
4.10 Schematic of the Intel Lab Berkeley Micra2Dot sensors . . . 52
4.11 Different humidity pattern code fuzzy membership functions with different boundary range overlapping sizes . . . 54
4.12 *acc* & *redu* results when varying the overlapping size of boundary data ranges . . . 54
4.13 *acc* & *redu* results with varying fuzzy deduplication thresholds . . . 55
4.14 Edge-level processing time when varying the number of data items . . . 55

5.1 Examples of sink node coverage areas, with multiple zones including hard and soft separations . . . 59
5.2 FREDD's sink-level deduplication process . . . 61
5.3 Basic spatial topological relationships following [14, 67] . . . 61
5.4 Deduplication quality metrics obtained with varying fuzzy deduplication thresholds . . . 64
5.5 Sample sink zone granularities using an extract of our reference Micra2Dot dataset (cf. Figure 4.10) . . . 65
5.6 *acc* and *redu* results with varying zone granularities considering static edge nodes . . . 66
5.7 Deduplication quality metrics obtained with varying zone granularities considering mobile edge nodes . . . 66

5.8 Deduplication quality metrics applied on sink node data (dataset 3), when varying the number of devices per sink zone 67
5.9 Sample sink zone use cases with sensor coverage area radius variations using an extract of the Micra2Dot dataset 68
5.10 Deduplication quality metrics applied on sink node data (dataset 3), when varying sensor coverage area radius with multiple hard/soft separation use cases 69
5.11 FREDD's time performance consider edge-and-sink level deduplication processes with varying parameters 70
5.12 Comparison of the deduplication quality metrics between RED, DRMF_1, DRMF_2, and FREDD, when varying the number of data measurements of dataset1 71
5.13 Percentage of deduplicates when fixing the first humidity data to $39.5\mu g/m^3$ and varying the second between $[37, 42]\mu g/m^3$ 71
5.14 FREDD's time performance compared with its recent alternatives, considering a fixed data size of 1000 items per edge, and a fixed number of 10 edge devices per sink node 72

6.1 Sample User Request Rule 77
6.2 Linking Redundancy Detection & Cleaning - Overview 77
6.3 Linking Redundancy Detection & Cleaning - Detailed 78
6.5 Deduplication Result - Consumer Request 1 .vs. Auto-Clean 79
6.6 Deduplication Result - Consumer Request 2 .vs. Auto-Clean 80

A.1 Fuzzification of Temperature $DataItem1$ and $DataItem2$ 84
A.2 Inference - Rule4 85
A.3 Inference - Rule5 85
A.4 Aggregation 86
A.5 Defuzzification 86

List of Tables

2.1 Data Redundancy Detection Comparison 17
2.2 Data Redundancy Cleaning Comparison 21

3.1 Data Items Example 25
3.2 Humidity Data Collection 28
3.3 Temperature Data Collection 28
3.4 Temporal Complexity Comparison 34

4.1 Device Data Items 41
4.2 Humidity Data Collection 41
4.3 Temperature Data Collection 41
4.4 Sample Disjoint Value Lookup Tables 42
4.5 Sample Intersecting Value Lookup Tables 42
4.6 Zone Lookup Tables 43
4.7 Pattern Codes - Humidity Data Collection 43
4.8 Pattern Codes - Temperature Data Collection 44
4.9 Filtering Output - Humidity Data Collection 45
4.10 Filtering Output - Temperature Data Collection 45
4.11 Result - Humidity Data Collection 51

5.1 Sink-level deduplication uses cases 59
5.2 Comparing FREDD with alternative solutions 70

6.1 Auto-Clean Mode (via Median) - Humidity Data Collection 75
6.2 Auto-Clean Mode (via Median) - Temperature Data Collection 75
6.3 Consumer-Centric Mode (via req_1 / p_1) - Humidity Data Collection . . . 77

A.1 Result - Temperature Data Collection 87

List of Abbreviations

IoT	**I**nternet **o**f **T**hings
HVAC	**H**eating **V**entilation **A**ir **C**onditioning
ICT	**I**nformation **C**ommunication **T**echnology
CE	**C**onnected **E**nvironment
M2M	**M**achine 2 **M**achine
NI	**N**etwork **I**nfrastructure
CI	**C**loud **I**nfrastructure
MEMS	**M**icro-**E**lectro-**M**echanical **S**ystems
SN	**S**ensor **N**etworks
WSN	**W**ireless **S**ensor **N**etworks
DRMCE	**D**ata **R**edundancy **M**anagement for **C**onnected **E**nvironments
DRMF	**D**ata **R**edundancy **M**anagement **F**ramework
FL	**F**uzzy **L**ogic
FREDD	**F**ramework for Fuzzy **R**edundancy **E**limination for **D**ata **D**eduplication
PWSNs	**P**eriodic **W**ireless **S**ensor **N**etworks
HDC	**H**istogram based **D**ata **C**lustering
EDC	**E**rror-aware **D**ata **C**lustering
RODS	**R**ecursive **O**utlier **D**etection and **S**moothing
V-RODS	**v**erification **R**ecursive **O**utlier **D**etection and **S**moothing
DMLDA	**D**ynamical **M**essaging **L**ist based **D**ata **A**ggregation
PSN	**P**eriodic **S**ensor **N**etwork
CCS	**C**luster based **C**ompressive **S**ensing Data Collection
CDC	**C**ompressive **D**ata **C**ollection
CS	**C**ompressive **S**ensing
RFID	**R**adio **F**requency **I**dentification
LMS	**L**east **M**ean **S**quare
CWCA	**C**ontrolled **W**indow-sized baed **C**hunking **A**lgorithm
ACA	**A**daptive **C**hunking **A**lgorithm
BKV	**B**locking **K**ey **V**alue
SVM	**S**upport **V**ector **M**achine
REDA	**R**edundancy **E**limination **D**ata **A**ggregation

Chapter 1

Introduction

1.1 Connected Environments, IoT, & Sensor Networks

In recent years, connecting physical devices to the Internet has been growing at an unprecedented rate. This phenomena has been fueled by significant technological advances in the field of IoT (Internet of Things) [1]. An example of connected objects includes thermostats and HVAC monitoring and control systems that manage Heating, Ventilation, and Air Conditioning in smart homes. IoT has played a remarkable role in other domains and environments to improve the quality of our lives. These applications range from transportation, to healthcare, industrial automation, and emergency response to natural and man-made disasters, where human decision making is difficult and requires assistance and input provided by data. Moreover, connected devices, and developments in ICT (Information and Communication Technology) have transformed traditional physical infrastructures (e.g., buildings, cities, hospitals, parking lots) into smart Connected Environments (CEs).

CEs host physical objects equipped with sensors capable of surveying the real-world and recording observations. Connected objects are considered data sources as well as data consumers since they are capable of exchanging data with other objects. CEs can host other data consumers as well such as data processing services, storage repositories/databases, and human users that query the environment. Therefore, CEs constitute a dynamic hub for data generation and consumption. This enables various application domains (e.g., energy management, healthcare, environment conservation) to benefit from the exploitation of the generated data and to provide several applications for the end users (e.g., energy management systems for smart buildings, patient monitoring in smart hospitals and elderly homes, pollution monitoring in smart cities). Figure 1.1 illustrates some domain specific applications that benefit from a CE.

FIGURE 1.1: CE impact on various application domains

The CEs offer equipment manufacturers, Internet service providers, and application developers a remarkable market opportunity. By the end of 2020, IoT smart objects reached 212 billion deployed entities in the whole world [2]. The rate of M2M (Machine2Machine) traffic flows is expected to take up to 45% of the whole Internet traffic, by the end of 2022 [2, 3, 4].

The impact of IoT-based services is also beneficial for the economic growth of businesses. The biggest economic impact is projected to come from Healthcare and manufacturing applications. IoT-based healthcare applications (e.g., mobile health, m-Health, and telecare) provide the means for medical wellness, disease prevention, diagnosis, treatment, and monitoring. The aforementioned applications are expected to generate approximately \$1.1–\$2.5 trillion worth of growth annually, in the global economy, by the end of 2025. The IoT is expected to cause an annual worldwide economic impact in the range \$2.7 trillion to \$6.2 trillion by 2025 [5]. The projected market share of dominant IoT applications is represented in Figure 1.2 [5].

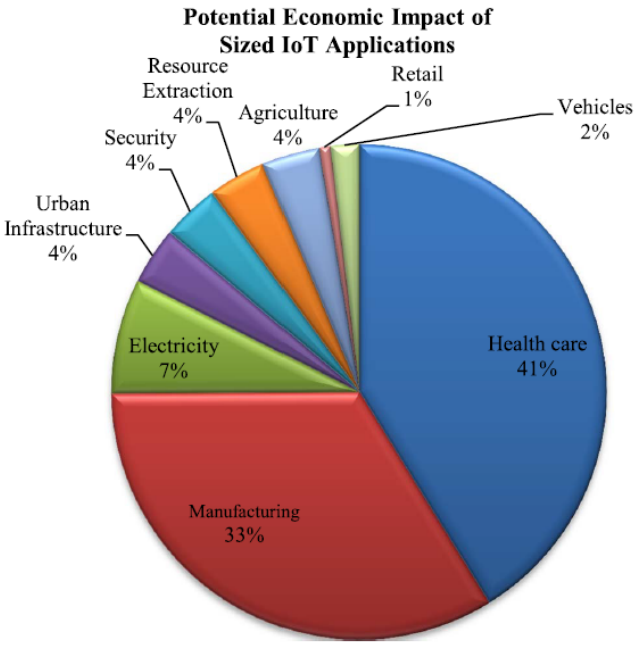


FIGURE 1.2: IoT projected market share by 2025

1.1.1 Key Elements in IoT

Key elements in IoT include: (i) Things; (ii) Gateways; (iii) Network Infrastructures (NI); and (iv) Cloud Infrastructures (CI) [6].

Things

Things represent the devices/sensors that sense stimuli from the environment and provide measurement data of the observations. The advances in micro-electro-mechanical systems (MEMS) technology, wireless communications, and digital electronics have paved the way for developing low-cost, low-power, multi-functional small-size sensor nodes [7]. These sensors could be deployed in different environments (e.g., buildings, cities) and/or embedded on various machines, equipment, and electronic platforms (e.g., mobile phones). Their advanced capabilities (e.g., sensing various properties, data transmission, and storing observations), increased autonomy (e.g., longer life cycles, more battery power, more fault/breakdown resistance), and miniaturization have allowed sensor networks to be widely adopted for environment monitoring. This has greatly benefited the proliferation of CEs.

Gateways

Gateways serve as an intermediate block between the things and network or cloud infrastructure. They ensure connectivity between things and between things and clouds. Things monitor the environment and report their observation to gateways (e.g., sink node, base stations etc.). These various gateways process data/information and forward these data to cloud infrastructure for storage purposes [8].

Network Infrastructures

The network infrastructure (NI) defines the topology, how sensors are deployed, where sensors are located, and to which sink each sensor reports its observation-s/sensed data. These infrastructures are used to manage and track the information flow within the environment. They also provide means for data processing, traffic management, and data security mechanisms [8].

Cloud Infrastructures

Cloud Infrastructures (CI) are equipped with data storage and are proficient in computing the data/information. These include a set of virtual servers, and storage repositories that are clustered together along with gateways and things. CI provide computing and analytical capabilities (various intelligent services, used to analyze information from various perspectives) [8].

1.1.2 Sensor Networks

The major components of CEs are Sensor Networks (SN). Wireless Sensor Networks (WSN) have emerged in the last few decades due to the significant technological advances in wireless communication, embedded computing techniques, and micro-electronic devices [9]. WSN group sensor nodes that are spatially dispersed and interconnected by using wireless communication [7]. These sensor nodes monitor the environment in which they are deployed (e.g., temperature and humidity) and produce measurement data values. We denote them as "Edge Devices" since they are deployed on the leaf, extremity, or edge of the CE. Edge device data is then sent to base stations for further processing/storage. We denote these base stations as "Sink Nodes/Devices" and locate them at the center or core of the network in an intermediary layer between the leaf layer (where we find the data sources, i.e., sensing edge devices) and the supervision/control layer (where we find the various data consumers). Figure 1.3 depicts a simplified representation of a CE network.

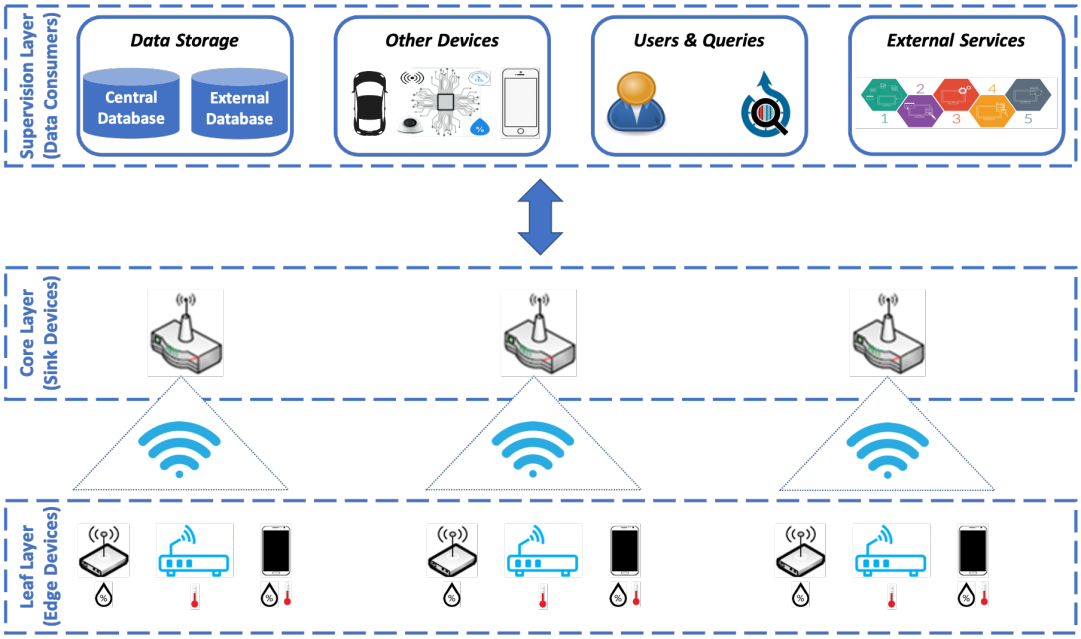


FIGURE 1.3: Connected Environment Network Representation

The sensors/devices in CEs continuously monitor physical environments (e.g., buildings, homes and cities) and report their observations to base stations. These devices/sensors rapidly produce huge amounts of data, as the sampling frequency can range from micro-seconds to hours. However, they are limited in terms of energy, processing and memory, and suffer from various inconsistencies (e.g., missing values, anomalies and redundancies).

1.2 Overview of Data Inconsistencies

This section provides an overview of the most common data inconsistencies in CEs. CE devices/sensors produce huge amounts of data, and send the data to base stations or other nodes continuously. However, the raw data from these devices suffer from various inconsistencies. We provide a brief overview of some of these inconsistencies.

1.2.1 Missing Values

In CEs, sensors may produce missing values due to various reasons. For instance, network issues such as unstable connectivity, communication bugs, packet loss, and congested network media could cause values to go missing during transmission. Moreover, hardware and autonomy issues (e.g., low battery power, equipment failures) could render some sensors unreliable and cause missing values. Therefore, missing data has been the focus of various research works regarding data refinement in CEs [10, 11, 12, 13, 14, 15].

1.2.2 Anomalies

Besides missing values, sensor data can suffer from anomalies. An anomaly is defined as an abnormal behaviour from the sequence of data generated by sensors/devices. The anomalies can be: (i) Point Anomaly; (ii) Contextual Anomaly; and (iii) Collective Anomaly [16].

- Point Anomaly: is an observation value that lies far from the rest of the data, also known as "outlier" [17].
- Contextual Anomaly: is an observation value that is normal in one scenario and abnormal in another. This type of anomaly requires the knowledge of context. It is also called conditional anomaly [18].
- Collective Anomaly: is an abnormal sequence of observations that are analyzed to find out the collective behavior of the data stream. For example, when determining the heart behavior in a healthcare application, a single observation at a time interval is not sufficient. Such cases require collective signals to determine whether heart behavior is normal or not [19].

1.2.3 Data Redundancies

Devices/sensors in CEs are often densely deployed to monitor the environment and report observations. These devices generate huge amounts of redundant data by sensing the environment. A redundancy produced by a device/sensor can be: (i) temporal; (ii) spatial; or (iii) spatial-temporal [20].

- Temporal Redundancy: this type of redundancy is caused by a single sensor producing similar observations at different time stamps.
- Spatial Redundancy: this type of redundancy is caused by similar sensors deployed in close vicinity to each other, thus producing redundant observations.
- Spatial-Temporal Redundancy: this type of redundancy is produced by mobile sensors/devices. Mobile sensors have the capability to move around the network. This might lead to generating redundant observations at different time stamps and locations simultaneously.

1.3 Thesis Context

In this thesis, we consider CEs from the perspective of data management. Specifically, we aim to provide CE users (e.g., smart building occupants, smart city managers) help in managing redundant data (e.g., detection and elimination) produced by various sensors/devices.

1.3.1 Objectives

To do so, we target the following specific objectives:

- Detecting redundancies at the source (i.e., on the edge devices/sensors), thus allowing the management of atomic or individual data redundancies (i.e., redundancies coming from one data source). This entails providing a data redundancy detection mechanism that runs on the edge of the network in order to flag redundant data prior to M2M communications and data storage at the center of the network (i.e., via the sink devices).
- Detecting redundancies at the core of the network (i.e., on the sink devices), thus allowing the detection of redundancies that appear from the combination of data coming from different sources (i.e., edge devices). This entails taking into consideration the physical and environmental constraints of the CE (e.g., physical separations between zones/areas like walls and windows and their impact on sensor coverage areas, device mobility and its impact on spatial-temporal provenance of data).
- Cleaning or removing the detected redundancies at the edge and at the core of the network (i.e., on the devices directly, and on sink nodes). We aim to introduce here a data consumer centric deduplication process that takes into account the consumers' needs and requirements for the redundancy cleaning phase (since the aforementioned needs might not be the same for all data consumers).

1.3.2 Motivation Scenario

We provide a CE example to illustrate the motivations of this thesis. Figure 1.4 shows a smart hospital, having two wards. The left ward describes a location map having a nursery, a drug storage room, and two ICU (Intensive Care Unit) rooms. Since these are critical areas of the hospital and require specific environmental constraints, they are separated by walls (physical hard separations). The right ward is dedicated for medical staff offices. Since no patients are hosted in this ward the constraints are more flexible, the offices are not hard-separated but consist of cubicles partially

open to one another (i.e., soft separations). In both wards, a set of static and mobile sensing devices (e.g., medical staff tablets) are deployed to monitor the environment. More specifically, the aforementioned sensors capture humidity and temperature measurements (among others) from their surroundings. Moreover, every ward has its own sink node that receives all data from the sensors located within its premises. Expert users (e.g., health and safety experts) require the data to regulated air quality and indoor temperature in the hospital to prevent issues such as overheating in the drug storage room, or bad air quality for patients in the ICU or babies in the nursery. The described setup is prone to generating a huge amount of data redundancies. Therefore, the following needs should be considered:

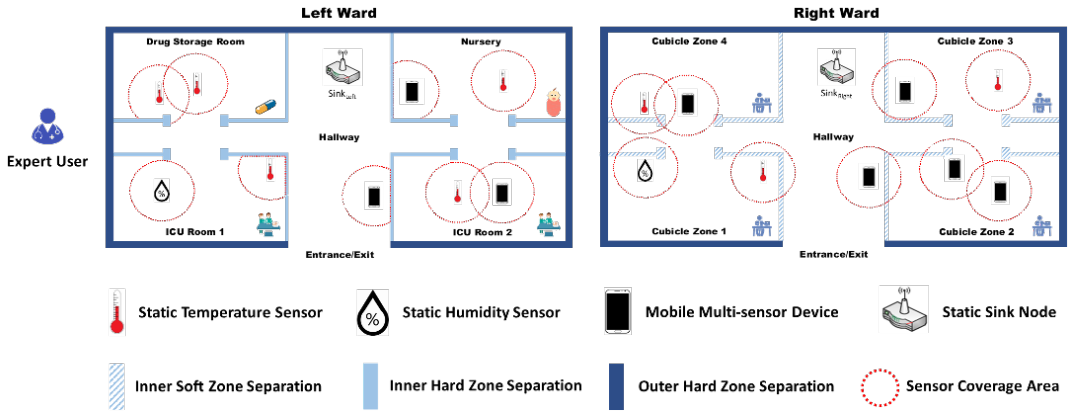


FIGURE 1.4: Smart Hospital Example

Need 1. Retrieving non-redundant and concise temperature/humidity data from individual locations in each ward in the hospital (i.e., querying data from static and mobile devices directly).

Need 2. Retrieving global non-redundant and concise temperature/humidity data from entire zones/wards in the hospital (i.e., querying data from the sink nodes directly).

Need 3. Defining health-related requirements for each area of the hospital (e.g., significant value deviations for temperature in the drug storage room, alarming humidity variations in the ICU) in order to enable data retrieval and storage based on the user's specifications.

In this setup, if humidity is stable in a specific room, multiple unnecessary duplicate values are sent to the sink and retrieved by the expert user. However, measurements made by the same mobile sensor moving between different locations, or measurements made by different sensors at different locations in the hospital might not be considered as redundant (e.g., what is not redundant at room level might be redundant at ward level). Therefore, data deduplication could be used to address the aforementioned needs if the following challenges are considered:

Challenge 1. How to accurately detect data redundancies at the static and mobile device level? How to consider spatial-temporal information to cope with device mobility? (cf. Need 1).

Challenge 2. How to accurately detect data redundancies at the sink level? How to consider the physical constraints of the environment (e.g., spatial constraints,

soft and hard zone separations, and sensor coverage areas) when detecting redundancies at the sink level? (cf. Need 2).

Challenge 3. How to allow expert users to configure the deduplication process in order to inject domain insights and knowledge for a more adapted and accurate redundancy detection and cleaning? (cf. Need 3).

There are several other challenges that emerge while considering data redundancies in connected environments (e.g., energy management, bandwidth management, redundant event detection). However, we only focus on the aforementioned needs and challenges (that will be detailed in following chapters). Next, we provide the details of the proposed framework and discuss how each module addresses a specific challenge.

1.4 Proposal

We present a brief overview of our proposal for Data Redundancy Mangement in Connected Environments, denoted as DRMCE. Our proposal addresses the objectives, needs, and challenges described in Section 1.3. Figure 1.5, depicts an overview of DRMCE's main modules. In the described process, the user decides if he/she wishes to handle redundancies directly at the edge device level (network source), at the sink device level (network core), or on both edge and sink levels. Regardless of where the deduplication is required, the process always consists of two main steps: (i) redundancy detection; and (ii) redundancy cleaning.

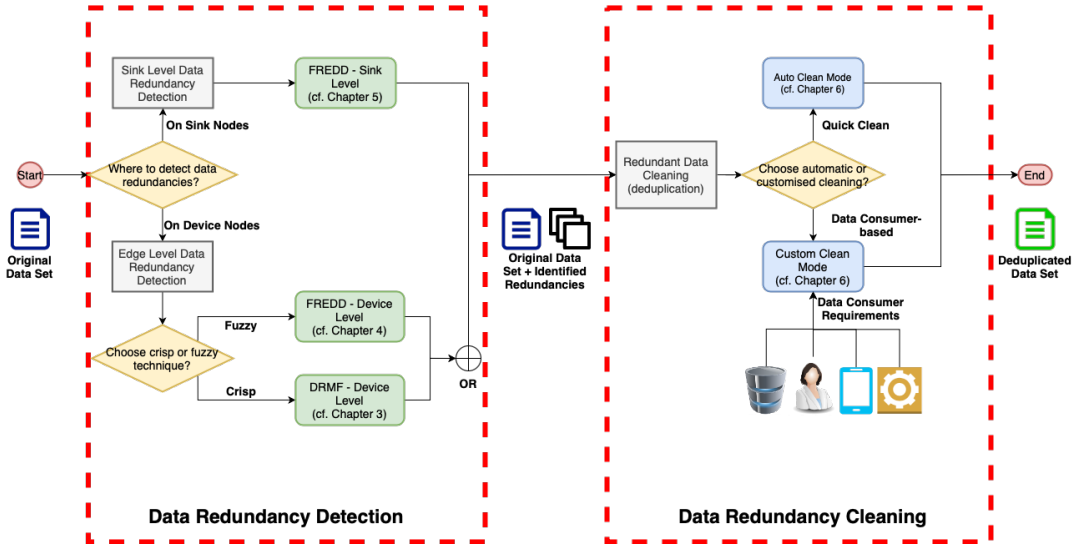


FIGURE 1.5: DRMCE Framework

1.4.1 Redundancy Detection at the Edge

In this thesis, we provide two approaches for redundancy detection at the edge device level. The first, denoted DRMF (Data Redundancy Management for leaF-edges), is based on clustering redundant data locally on devices using crisp thresholds. The second approach, denoted FREDD (Fuzzy Redundancy Elimination for Data Deduplication), improves on the first proposal by introducing fuzzy reasoning to the redundancy detection process in order to address the shortcomings of the

crisp thresholds that sometimes generate imprecision. These two approaches enable redundancy detection on static and mobile edge devices (cf. Challenge 1).

1.4.2 Redundancy Detection at the Sink

Moreover, we provide an extension of FREDD to cover redundancy detection at the sink level (i.e., at the core of the network). To do so, we take into consideration features such as the spatial constraints of the environment (e.g., locations, location maps), physical elements (e.g., hard and soft zone separations), sensor properties (e.g., coverage areas) in addition to the features already considered in the FREDD proposal at the edge device level (e.g., mobility). This allows accurate detection of redundancies at the sink level (cf. Challenge 2).

1.4.3 Redundancy Cleaning

Finally, once redundancies are detected, the process moves into the redundancy cleaning phase. In this work, we present an auto-clean mode that provides traditional cleaning mechanisms (e.g., summarizing a redundant set of data to one representative centroid value such as the mean, median, min, or max). In addition, we provide a consumer-centric cleaning mode that can be tailored, configured, and customized to fit the needs of data consumers including databases, human users, other devices, and data processing services (cf. Challenge 3).

1.5 Report Organization

The rest of the thesis is organized as follows:

Chapter 2 reviews the state of the art on data redundancy management. We go through some of the existing approaches applied in data storage and data warehousing solutions. Then, we focus on data redundancy detection works dedicated to connected environments. This includes works that target the edge level of the network, and approaches that target the sink level. Finally, we review data redundancy elimination/cleaning approaches. This chapter also includes a comparative study of these works based on a proposed set of criteria.

Chapter 3 describes the Data Redundancy Management for leaF-edges (DRMF) for detecting data redundancies at the device level. In order to cope with mobility, we define two types of data redundancies: (i) temporal redundancies (to be detected on static devices); and (ii) spatial-temporal redundancies (to be detected on mobile devices). We propose a redundancy detection algorithm for each of the aforementioned redundancies. The algorithms consist of evaluating the deviations between successive data values generated by a device using a specific variation threshold in order to cluster redundant subsets of data. Finally, we evaluate the performance of DRMF as well as its accuracy when detecting redundancies.

Chapter 4 describes the Fuzzy Redundancy Elimination for Data Deduplication framework (FREDD) for detecting data redundancies at the device level. FREDD improves our initial proposal (DRMF) by overcoming the limitation of having crisp thresholds that can lead to potential drops in precision when detecting redundancies. This chapter provides a background on fuzzy reasoning, presents the new framework, details its modules, and explains how one can use it to detect

data redundancies locally on static and mobile edge devices. Finally, we include a extensive experimentation of FREDD's performance and accuracy.

Chapter 5 describes an extension of FREDD to address data redundancy detection challenges at the network sink level. We consider a variety of environmental, spatial, physical, and device-related features that affect redundancy detection at the core of the network. We consider locations, inter-location hard/soft separations, and sensor coverage areas. Finally, we provide details of the experiments and results.

Chapter 6 discusses the elimination/cleaning of previously detected data redundancies. DRMCE provides two main modes for cleaning redundant data: (i) the auto-clean mode where we automatically summarize a set of redundant data to one representative data item; and (ii) the data consumer-centric mode where we consider the requirements of data consumers to tailor the cleaning process accordingly. We also discuss how the data consumer requirements can be taken into account in the cleaning process. Finally, we present the experimental evaluation and results.

Chapter 7 concludes the report by providing a recap of all aforementioned chapters, and discusses the next steps and potential future directions.

Chapter 2

Related Works

We review here existing approaches for data redundancy detection and cleaning. We discuss first the topic of deduplication in data storage and warehousing solutions. Then, we shift our focus towards CEs starting with data redundancy detection and then data redundancy cleaning approaches.

2.1 Deduplication in Data Storage and Data Warehousing

The automatic removal of duplicate data tokens has been primarily used in archival and backup systems (e.g., Microsoft Farsite, HYDRAstore, DEBAR), primary storage (e.g., Microsoft Windows Server, Oracle ZFS), RAM (e.g., VMWare ESX, Linux KSM), and SSDs (e.g., Cache Acceleration Software CAS by CAFTL) [21]. They mostly rely on chunk-level deduplication which splits the incoming data into multiple chunks, and generates a unique hash value for every individual chunk, referred to as the chunk's fingerprint [22]. Deduplication is then performed by eliminating the chunks having identical fingerprints. Among the many chunking algorithms are Rabin fingerprinting algorithm, TD (Two Divisors) algorithm, TTTD (Two Thresholds, Two Divisors) algorithm, and MAXP algorithm [22, 23]. Data deduplication is also a necessary step in data cleaning, also referred to as data scrubbing in data warehousing [24]. It consists in matching data records that relate to the same entities from several databases. Many techniques have been used in this context, including correlated sub-queries, temporary tables, derived tables, Common Table Expressions (CTEs), and dynamic SQL [25]. Most of these techniques are deterministic and require a unique entity identifier (or key) available across all the records/databases to be linked, or for all the databases to have the same structure. Some of them also consist of holding all distinct records in temporary or new tables which require big storage space. One major issue with the latter techniques is the time overhead needed to perform the extensive comparison operations between data records. More recent approaches aim at reducing data record comparison time by performing a pre-processing indexing step where each record is assigned a Blocking Key Value (BKV), and then records having the same or similar BKVs are clustered and compared together [24]. Some of the used clustering techniques include Sorted Neighborhood, Q-gram based clustering, and Canopy clustering [24]. Finally, data deduplication is applied to storage backup systems with the aim of reducing storage costs and improve storage space utilization [26]. This approach divides a source data S into a set of chunks, $C = \{c_1, c_2, ..., c_n\}$, by using a chunking algorithm. If data changes in S, then it should be denoted as S'. This changed data is divided into chunks, $C' = \{c'_1, c'_2, ..., c'_n\}$. These two chunks are then compared and only unique chunks are stored again in repository. Data deduplication is applied in two ways. In first way, data is first stored and then redundant data is removed. While in second method [27], data deduplication is applied on online data in real

time.

Discussion: Most deduplication techniques for data storage and data warehousing assume textual data duplicates only and disregard numerical values, e.g., [22, 23, 24]. The few methods which address numerical data like the proposal in [25] assume exact duplicates (e.g., exact temperature measurements) and disregard approximations (e.g., 15 °C and 15.1 °C are considered as different tokens). However, numerical tokens are of central importance in connected environments, where most data collected from sensors are scalar.

2.2 Redundancy Detection in Connected Environments

2.2.1 Redundancy Detection at the Network Edge

The devices/sensors in connected environments sense the real world state and send data to sink or base station nodes. The devices found in such environments can be static or mobile. Static devices do not change their position, while mobile devices move around in the environment. However, the environment often remains stable for extended periods of time and these devices/sensors produce similar observations. For example a temperature sensor deployed in a building or outside in open areas. They send these observations to upper layers of the network which leads to energy depletion and storage overhead at central storage places or databases. In the literature, several authors proposed strategies in order to target redundant data at device/sensor level. Below are some of the techniques which handle redundant data at device/sensor level.

The authors in [28] proposed an approach for cleansing indoor RFID data. They focused on two tasks; temporal redundancy elimination and spatial ambiguity reduction. For detecting temporally redundant data, they group redundant data that belong to a temporal interval. To tackle spatial ambiguity, they propose a distance graph approach to capture spatial-temporal deployment constraints of RFID readers. They exploit these constraints to detect redundant data within a spatial-temporal coverage. Although this work handles data redundancies at device level, it does not consider the dynamicity of the devices and environment for spatial-temporal redundancy detection.

The approach described in [29] focuses on managing redundant content in multimedia data (e.g images/videos uploaded in constrained based wireless networks). The strategy is based on similarity of metadata (i.e. image features) between an incoming image and an already stored one. It focuses on reducing latency in network. However, the approach only considers images uploaded from mobile devices and does not consider redundancies in images from static devices.

Harb H. et.al, in their approach [30], focused on filtering data at sensor layer in PSNs (Periodic Sensor Networks). They filter data using Pearson coefficient metric on sensor nodes before forwarding to an aggregation node. However, they do not consider mobility of devices while considering redundancy at device level.

In [31], the authors present a data reduction scheme for IoT (Internet of Things) using in-networking data filtering. This approach filters out redundant data at sensor nodes before forwarding it to sink nodes. Data filtration is based on data change detection and deviation of observed values from estimated values. Though this approach detects data redundancies from edge sources, it does not consider mobile devices while detecting redundancies.

In [9], a strategy for two-tier data reduction is proposed for connected environments. This approach works on the sensor layer and gateway layer of the network. At the sensor layer, authors used simple compression techniques to reduce huge numbers of data produced by sensors/devices. They exploit temporal correlation among the data for reducing data before transmitting to sink or gateway nodes. Although this approach reduces the number of transmissions from sensors to gateways, it ignores the mobility of devices at source layer. Another approach discussed in [32] detects redundant data at the source nodes as well as at the sink nodes. At the source nodes they used the Least Mean Square (LMS) metric to estimate the next value produced. If the actual value is within deviation threshold, then it is identified as a redundant value. The same method is applied at the sink node.

The approach presented in [33] focuses on compressing time series data for Internet of Things (IoT). This approach is applied on the devices in a connected environment. In order to detect redundant data, the authors identify correlation between device data by finding variance in the values. The variance is based on a deviation threshold. If the variance is within a deviation threshold, then those values are considered redundant.

In [34], the authors put forward a novel technique denoted DiDAMoK (Distributed Data Aggregation based Modified K-Means) for redundancy detection in sensed data found IoT environments hosting PWSNs (Periodic Wireless Sensor networks where data is produced in periods). This proposal runs in three stages: (i) locally storing sensor observations every period; (ii) clustering the observations using a modified version of K-means; and (iii) forwarding one representative observation per redundant data cluster to a sink node. This approach aggregates data on static devices to remove duplicates and does not consider device mobility.

Finally, the authors in [35] propose a data redundancy management technique using an unsupervised learning approach based on data clustering. The authors suggest clustering edge nodes based on their produced sensory data in order to aggregate identical data to eliminate redundancies, before storing the data in the cloud. However, they do not consider device mobility and spatial-temporal redundancies.

2.2.2 Redundancy Detection at the Network Sink

To avoid exhausting the often limited resources of edge devices, several works delegate the redundancy management to the network core where more powerful equipment are often found (e.g., aggregation, sink, or base station nodes). Network core nodes normally have sufficient processing and memory resources. Mostly, data aggregation techniques are considered, for processing redundancy of data at this level. Data aggregation considers various diverse data sources. It gathers these sources and builds an accurate phenomena under observation. In IoT and WSN, various data aggregation techniques are found. These include operations like sum, maximum, minimum, count etc. Aggregation methods can be divided into centralized and distributed schemes. Centralized methods rely on continuous communication among network nodes leading to additional costs. This type of method is not suitable for sensor networks. This is the reason, distributed methods such as clustering, multi-path and aggregated trees are becoming adopted as well [36].

In [37], a data aggregation based scheme was proposed for reducing the size of data. The scheme is called Prefix-Frequency Filtering (PFF). This approach has two aggregation layers, first layer is on the sensor node while the second is on cluster

head level. The authors use the Jaccard similarity measure for filtering redundant data on both layers. This approach does not consider redundancies produced by mobile devices.

Another approach based on data aggregation is presented in [38]. The authors introduced an approach called Dynamical Message List based Data Aggregation (DMLDA). The technique is based on dynamic list data structure. They store historically received measurements and then apply redundant data filtration. However, this technique only focuses on data redundancies from the perspective of static devices.

Similarly in [39], the authors divide sensors into non-overlapping regions. They sample and apply aggregations to data from these regions. Most data aggregation schemes use trees or other fixed data structures. The authors however, do not consider sensor mobility in order to detect spatial-temporal redundancies.

In [40], the authors present an approach based on spanning tree. In this scheme, every leaf node senses data from a specified location. Then, data aggregation is applied starting from the leaves and propagating towards the root. This scheme only focuses on handling redundancies coming from static devices.

In [41], the authors proposed a distance based data aggregation scheme for eliminating data redundancy in PSNs (Periodic Sensor Networks). They consider PSN as a form of connected environment which produces observations in periods. Data aggregation is performed on data at sensor node and cluster head level before transmitting it to sink node or base station. They evaluated their approach on four distance measures: (i) Euclidean Distance; (ii) Bray Curtis Distance; (iii) Cosine Distance; and (iv) Canberra Distance. However, the authors did not take into account, the environmental features of the CE nor the mobility of devices.

There are some approaches that use data compression techniques to handle data redundancies in connected environments. The aim of compression techniques is to reduce the amount of data routed through the network [42, 43]. Data compression techniques focus on compressing data before transmitting it to upper level nodes in the hierarchy. The approach proposed in [43] focuses on compressing data in sensor networks that are organized into sensor clusters. The approach, denoted Cluster Based Compressive Sensing Data Collection (CCS), performs data compression at cluster head level by generating Compressive Sensing (CS) measurements based on block diagonal matrices created from raw data received from neighbouring sensors. Furthermore, the authors reconstruct CS measurements at the base station (Sink). However, this work does not consider dynamic redundancies produced by mobile devices.

The authors in [44] propose a compression-based technique called Compressive Data Collection (CDC). This scheme focuses the spatial-temporal correlation to achieve data compression. It consists of two layers. These layers are, opportunistic routing with compression and nonuniform random projection based estimation for reconstruction. They only focused on redundancies generated by static devices.

The authors in [45], propose an approach for handling data redundancy on sink nodes. They considered static devices that produced and sent data to sink nodes for further processing. The sink nodes apply a data aggregation mechanism on the received data and remove redundant data produced by sensors in clusters. Tha approach detects both intra and inter cluster data redundancies. Intra cluster redundancies are addressed on cluster heads while inter cluster redundancies are managed on sink node. The overall objective of the authors is to efficiently utilize the bandwidth of the network.

In [46], the authors propose an approach for data reduction in connected environments. They detect redundant data at the IoT-gateway layer. In their approach they use primary and secondary gateways for processing redundant data concurrently. To achieve this, the approach divides the sensed data into odd and even records. Odd records are processed by a primary gateway and even records are processed by a secondary gateway. The primary purpose of this division was to avoid single point of failure at the network core. Furthermore, they only considered redundancies based on static devices and did not consider the environmental and physical features of the CE.

The approach discussed in [47] focuses on detecting data redundancies at two levels (i.e., at edge device and sink levels). The authors focus on identifying temporal and spatial redundancy in case of static devices. In order to detect a temporal redundancy, they calculate deviations between values at current and previous timestamps using Kalman filter. If the values are within a specified threshold, then they are considered duplicates. For detecting spatial redundancies, the authors group sensor nodes based on locations and sensing ranges. If the location and sensing ranges overlap, then those nodes are declared as redundant nodes. However, the authors do not take into account sensor mobility.

In [48], the authors propose an approach for detecting spatial data redundancies. They rely on a correlation tree of data location stamps and sensing ranges of sensor nodes. The correlation tree reveals spatial correlation between sensors using Euclidean distance. Then, the authors declare data as redundant if the Euclidean distance of two or more nodes is within a specified deviation threshold. However, physical environmental features of CE and sensor mobility are not considered.

In [49], the authors propose a clustering-based appraoch for data redundancy detection. More specifically, this approach uses Histogram based Data Clustering (HDC) to detect clusters of redundant data. The authors applied data clustering at the cluster head level.

The authors in [50] focus on the spatial distribution of sensors in CE to prevent deployment overlaps that could lead to redundant data. To do so, a graph of nodes and detected events is constructed from raw sensory data to identify nodes producing redundant data. The proposal does not consider sensor mobility.

The approach presented in [51] aims to maximize the lifetime of the network in connected environments. To do so, the authors use a clustering technique based on heuristics to detect clusters of redundant data. They form clusters of sensors using proximity based values.

The authors in [52] propose an Error-aware Data Clustering Technique (EDC) that consists of three main modules: (i) Histogram Based data clustering; (ii) Recursive Outlier Detection and Smoothing (RODS); and (iii) verification of RODS (V-RODS). The user has a choice of picking one module at a time based on his requirements. The purpose of all three modules is to detect clusters of redundant data by analyzing temporal correlations. RODS with HDC is used for error-aware data clustering. This module identifies random outliers using the temporal correlation of data to maintain data reduction errors within an acceotable threshold. VRODS with HDC identifies not only random outliers but also frequent outliers based on both temporal and spatial correlations of data. The approach only works for static devices.

In [53], the authors address data redundancies at the core of the network using a supervised machine learning solution based on Support Vector Machines (SVM). They build an aggregation tree for the given size of the network and then apply SVM

to recognize data redundancies. The authors target temporal and spatial redundancies once the data is consolidated in a central node, which provides a redundancy-free data repository that can be mined using dedicated data processing techniques. However, redundancies are not handled at the edge level, and data exchange between devices at the edge remains costly due to unnecessary communications.

In [54], the authors provide a data deduplication technique in healthcare-based IoT by introducing a Controlled Window-size based Chunking Algorithm (CWCA) to identify cut-points in sensor data distributions. Yet similarly to [53], the solution in [54] only performs data deduplication at sink nodes and does consider redundancies at edge devices. Moreover, the solutions in [53, 54] do not consider spatial-temporal redundancies generated by mobile devices. Similarly, another approach [55] applies an Adaptive Chunking Algorithm (ACA) to identify cut-point between two windows. They apply their technique on the collector node (e.g., cluster head).

The authors in [56] propose a factorization-based technique. They represent sensory data using the SSN ontology. Then, the data is factorized using observation multiplicity and measurement multiplicity concepts. Observation multiplicity being the "count of the observations regarding an observed phenomena produced by a sensor having same values". While measurement multiplicity is defined as the "count of measurements having same values and unit of measurements". The aim of factorization is to reduce the multiplicity of observations and measurements from n to 1.

2.2.3 Redundancy Detection Comparison

To compare existing approaches, we propose the following criteria based on the needs and challenges discussed in Section 1.3.2:

- Criterion 1. Edge Redundancy Detection: stating if the approach detects data redundancies at the device level (i.e., at the source). This enables efficient querying of the edge devices as well as preventing unnecessary data from spreading throughout the network (cf. Need 1 - Challenge 1).
- Criterion 2. Core Redundancy Detection: denoting if the approach detects additional (composite) redundancies that appear on sink nodes (i.e., in the core of the network) due to data collection from various devices. This enables the detection of redundancies that don't appear on the source (cf. Need 2 - Challenge 2).
- Criterion 3. Dynamicity & Environment Consideration: specifying if the approach detects dynamic redundancies due to device mobility and the physical, spatial, and technical constraints of the environment (e.g., inter-zone separations) and the devices (e.g., coverage areas). This allows the redundancy detection to be more accurate by considering contextual features in addition to data value similarities (cf. Need 1,2 - Challenge 1,2).

Table 2.1 shows a comparative recap of the aforementioned approaches found in the literature. It also highlights the fact that none of these works fully covers all of our proposed criteria.

TABLE 2.1: Data Redundancy Detection Comparison

Approach	Criterion 1 Edge Redundancy	Criterion 2 Core Redundancy	Criterion 3 Dynamicity & Environment
Baba A. et al. [28]	✓	✗	✗
Dao T. et al. [29]	✓	✗	✓
Harb H. et al. [30]	✓	✗	✗
Idrees A. et al. [34]	✓	✗	✗
Ismael W. et al. [31]	✓	✗	✗
Qurabat A. et al. [9]	✓	✗	✗
Chowdhury S. et al. [50]	✗	✓	✗
Patil P. et al. [53]	✗	✓	✗
Ullah A. et al. [55]	✗	✓	✗
Ullah A. et al. [54]	✗	✓	✗
Li S. et al. [35]	✓	✗	✗
Alam M. K. et al. [49]	✗	✓	✗
Karim F. et al. [56]	✗	✓	✗
Makhoul A. et al. [37]	✓	✓	✗
Du et al. [38]	✗	✓	✗
Huang et al. [40]	✗	✓	✗
Nguyen et al. [43]	✗	✓	✗
Liu X. et al. [44]	✗	✓	✗
Mantri D. et al. [45]	✗	✓	✗
Dasgupta K. et al. [51]	✗	✓	✗
Ling S.W et al. [46]	✗	✓	✗
Alam M. K. et al. [52]	✗	✓	✗
Fathy Y. et al. [32]	✓	✗	✗
Yemeni Z. et al. [47]	✗	✓	✗
Ismael W. et al. [48]	✗	✓	✗
Blalock D. et al. [33]	✓	✗	✗
Song Lin et al. [39]	✗	✓	✗
Hassan Harb et al. [41]	✗	✓	✗

2.3 Data Redundancy Cleaning

2.3.1 Data Redundancy Cleaning at the Network Edge

The authors in [28] propose an approach for cleansing indoor RFID data. They focused on two tasks: (i) temporal redundancy elimination; and (ii) spatial ambiguity reduction. To eliminate temporal redundancies, they aggregate raw RFID readings within a temporal period without loss of information. For handling spatial ambiguity, they propose a distance graph approach to capture spatial-temporal constraints of RFID readers' deployment. They exploit these constraints to remove redundant data within spatial-temporal coverage among RFID data. Although this work removes redundant data from RFID devices it does not provide a configurable cleaning mechanism capable of considering specific redundancy cleaning requirements.

The authors in [52] propose an Error-aware Data clustering Technique (EDC). This technique consists of three modules: (i) Histogram Based data clustering; (ii) Recursive Outlier Detection and smoothing (RODS); and (iii) verification of RODS (V-RODS). The purpose of all three modules is to cluster data based on temporal interval and aggregate data into one value and send this value to other layers of the network. The user has the choice of picking one of the three modules for redundancy cleaning based on the requirements. Although this approach discusses a requirement-based data redundancy cleaning, it does not consider data consumer needs from a data consumption stand point.

In another similar approach [49], the authors propose an aggregation-based approach to eliminate redundancies from sensor data. They cluster redundant data within temporal periods and reduce each redundancy clusters to one representative value. At the end this non-redundant data is shared with sink node. This technique does not provide a configurable cleaning mechanism that could be personalized to adapt to data consumer needs. Similarly in [34], the authors propose an aggregation based approach to eliminate redundant data produced by sensors. They aggregate values, within a cluster, for a specified time period, using average function. These reduced values are forwarded to sink node.

The approach presented in [51] focuses on maximizing the lifetime of the network in connected environments. To do so, the authors use clustering based heuristics to gather data from sensors in the network and detect clusters of redundant data. After that, data is aggregated and sent to sink nodes for further processing. They form clusters of sensors using proximity based values. After that, maximum lifetime schedule is computed and a tree is constructed based on this schedule for each cluster. The authors, however, only focus on cleaning redundant data via an automated non configurable mode.

In [53], the authors use Locality Sensitive Hashing (LSH) to eliminate redundant data. The data produced by each sensor, in each session (i.e., time period), is encoded using LSH encoding. Each sensor sends LSH codes to aggregation nodes. The latter compare these codes to find similarities, then select only one sensor node, for forwarding data to sink or base station. However, this approach does not give importance to data and its usage by sink node or base station.

In [33], the authors propose SPRINTZ, a time series compression approach that compresses data generated by sensors in IoT environments. This approach forwards summaries of data in a compressed form to other layers of the network and central storage repositories. This approach compresses data in the following steps: (i) It employs a forecaster that predicts each sample based on previous samples. Then it predicts the difference between next sample and predicted sample; (ii) it bit packs the errors as "paylod" and adds a header having information to invert the bit-packing; (iii) it finds the run-length encoding of all non-zeros payload and writes out the number of payload; and (iv) it applies Huffmad coding to the headers and payload and forwards this data to other layers of the network. The other layers of the network decode the data using the information attached in the payload. This approach only compresses temporal data and ignores the spatial-temporal characteristics of data (for mobile devices). Furthermore, it does not consider requirements based compression while compressing the data.

In [48], the authors propose an approach for detecting and cleaning spatial data redundancy. They build Correlation Tree based on location and sensing range of sensor nodes. Correlation tree construction finds spatial correlation between sensors using Euclidean distance. After constructing tree, they apply a data fusion technique in order to clean redundant data produced by sensors in each other's coverage and sensing area. The authors clean redundant data using data fusion every time intermediate nodes receive data from sensor nodes, hence not taking into account requirements based data cleaning.

In [31], the authors handle redundancy cleaning by employing value based deviations at the source (i.e., sensor layer) and aggregation at the data fusion layer. The approach, caches first reading at a particular time, and then calculates the difference of cached reading, with upcoming value of the sensor. This layer only forwards data to data fusion layer, if there is significant deviation between the two values, otherwise data is discarded. The data fusion layer aggregates data within same time

domain, received from data filtering layer. At the end, the non-redundant data is forwarded to the cloud for storage purposes. This method doesn't consider what and how much data is required by the sink node or the cloud.

The approach presented in [30] focuses on filtering data at sensor layer in PSNs (Periodic Sensor Networks). It filters data using Pearson coefficient metric on sensor nodes before forwarding data to aggregation node. However, the authors do not consider cleaning based on some requirements of devices, central storage, or other services.

2.3.2 Data Redundancy Cleaning at the Network Sink

The approach discussed in [47] focuses on cleaning data redundancy at two levels i.e., at edge device and sink level. The authors, focus on eliminating temporal and spatial redundancy in case of static devices. In order to eliminate temporal redundancy, they calculate deviations between values at current and previous timestamps using Kalman filter. They only forward data that has significant deviation from the previous value. For cleaning spatial redundancy, authors group sensor nodes based on location and sensing ranges. After grouping similar nodes, they aggregate the values by finding mean of values and send one value per group to gateway layer. The authors clean redundant data based on value deviations (at edge devices) and periodically aggregate data (at sink layer) and forwards non-redundant data to the gateway layer. However, they do not consider eliminating/cleaning redundant data based on some requirements/needs between the core of the network and edge devices.

The approach presented in [35] clusters edge nodes based on spatial distributions. Within each cluster, the authors segment the sensed data into chunks. They eliminate redundancies among these chunks by finding unique chunks of values. After that, these unique chunks are forwarded to central cloud for storage purposes. This method finds unique chunks of data every time nodes produce data, while ignoring the importance of the data storage requirements of the central cloud.

In [54, 55], a deduplication based approach is proposed to eliminate data redundancies on data aggregation nodes. The devices attached to a patient body may produce similar readings within a short period of time. Instead of sending these similar values, this approach encodes each similar value to 'T' and sends 'T' to the aggregation node. Otherwise, it sends the actual reading. On the aggregation node, duplicates are removed before sending the redundancy-free data set to the cloud. This approach continuously performs redundancy cleaning in the same way regardless of the data consumer and its needs.

The deduplication technique proposed in [43] focuses on compressing data in sensor networks. The approach is called, Cluster Based Compressive Sensing Data Collection (CCS). It consists of compressing data at cluster head level by generating Compressive Sensing (CS) measurements based on block diagonal matrices created from raw data. Furthermore, the authors reconstruct CS measurements at the base station (Sink). This approach ignores data redundancy cleaning based on user or device based requirements.

The authors in [44] proposed compression based technique called Compressive Data Collection (CDC). This scheme focuses the spatial-temporal correlation to achieve data compression. It consists of two layers. These layers are, opportunistic routing with compression and nonuniform random projection based estimation for reconstruction. However, the authors do not clean redundant data based on data consumer requirements.

In [46], an approach for data reduction in connected environments is proposed. The authors propose to detect redundant data at the IoT-gateway layer. However, this technique does not consider personalized cleaning based on user, database, device, nor service requirements.

Another approach discussed in [32], detects redundant data at the source nodes as well as at the sink node. At the source node they used Least Mean Square (LMS) technique to estimate the next value produced. If the actual value is within deviation threshold, then it is identified as a redundant value. They forward values which do not fall in deviation threshold to other layers of the network.The same method is applied at the sink node.

The technique described in [56] eliminates data redundancies based on data factorization. To do so, it factorizes, observations and measurements multiplicity to one occurrence for similar observations, for each attribute (i.e., temperature, humidity etc.). This single occurrence is then forwarded to cloud where it is stored. Data redundancy cleaning is only based on exact similarity between observations, hence, does not consider small deviations in data and also surpasses eliminating redundant data based on context/requirements.

2.3.3 Data Redundancy Cleaning Comparison

To compare existing approaches, we propose here the following criteria based on the limitations discussed in previous subsection:

- Criterion 4. Edge Redundancy Cleaning: stating if the approach deduplicates data redundancies at the device level (i.e., at the source). This enables efficient querying of the edge devices as well as preventing unnecessary data from spreading throughout the network (cf. Need 1 - Challenge 1).
- Criterion 5. Core Redundancy Consideration: denoting if the approach cleans additional redundancies that appear on sink nodes (i.e., in the core of the network) due to data collection from various devices. This enables the deduplication of redundancies that don't appear on the source (cf. Need 2 - Challenge 2).
- Criterion 6. Consumer-based Cleaning: specifying if the approach considers data consumer needs and requirements when cleaning data redundancies (cf. Need 3 - Challenge 3).

Table 2.2, presents the comparison of related approaches based on the criteria defined above. The table shows that none of the approaches proposed in literature fully covers all the criteria.

2.4 Recap

In this chapter, we reviewed several approaches from a data redundancy detection and cleaning point of view. Most of the discussed approaches detect redundancies either at the core of the network (i.e., sink or base station level) or at the data storage level once data is stored in repositories. Fewer approaches focus on detecting redundant data at the device level. Moreover, most of the aforementioned works focus on data value similarity and none of them fully considers the contextual elements of a connected environment when detecting redundancies (e.g., physical,

TABLE 2.2: Data Redundancy Cleaning Comparison

Approach	Criterion 1 Edge Redundancy Cleaning	Criterion 2 Core Redundancy Cleaning	Criterion 3 Personalized Cleaning
Yemeni Z. et al. [47]	✓	✓	✗
Ismael W. et al. [48]	✓	✗	✗
Alam M. K. et al. [52]	✓	✗	✗
Alam M. K. et al. [49]	✓	✗	✗
Idrees A. et al. [34]	✓	✗	✗
Ismael W. et al. [31]	✓	✗	✗
Patil P. et al. [53]	✓	✗	✗
Li S. et al. [35]	✗	✓	✗
Ullah A. et al. [55]	✗	✓	✗
Ullah A. et al. [54]	✗	✓	✗
Karim F. et al. [56]	✗	✓	✗
Dasgupta K. et al. [51]	✓	✗	✗
Ling S.W. et al. [46]	✗	✓	✗
Liu X. et al. [44]	✗	✓	✗
Nguyen et al. [43]	✗	✓	✗
Blalock D. et al. [33]	✓	✗	✗
Harb H. et al. [30]	✓	✗	✗
Fathy Y. et al. [32]	✓	✓	✗
Asif Iqbal Baba et al. [28]	✓	✗	✗

spatial, and technical constraints). Furthermore, the proposed cleaning mechanisms have evolved around either deleting or summarizing redundancies in a generic and automated way without considering the different needs of data consumers in this process. Finally, besides discussing related approaches, we provided a comparative study based on predefined criteria which was based on the literature and the needs/challenges of this work.

Chapter 3

DRMF: Data Redundancy Management Framework

In this first contribution, we focus on detecting data redundancies in connected environments at the device level. This entails providing a flexible framework for data redundancy detection capable of handling the constraints of the environment and the mobility of devices. To do so, we propose DRMF (Data Redundancy Management for leaF-edge), a device-based approach capable of clustering redundant data locally on static/mobile devices (cf. Challenge 1).

In this chapter, we describe DRMF, formally define the redundancies that DRMF is capable of detecting, and detail its clustering-based algorithms. The latter help detect temporal and spatial-temporal redundancies in order to consider both static and mobile devices/sensors.

Thereafter, we evaluate DRMF by presenting our experimental protocol, the conducted experiments, and the obtained results. Finally, we discuss the results and highlight the limitations of DRMF and its possible extensions.

Elio Mansour, Faisal Shahzad, Joe Tekli, and Richard Chbeir. 2020. Data Redundancy Management in Connected Environments. In Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '20). Association for Computing Machinery, New York, NY, USA, 75–80. DOI:https://doi.org/10.1145/3416013.3426451
Elio Mansour, Faisal Shahzad, Joe Tekli, and Richard Chbeir. Data Redundancy Management Framework for Connected Environments. Submitted to the Computing Journal.

3.1 Introduction

In CEs, significant volumes of data are continuously produced, exchanged, and stored with the passage of time. This massive amount of data is the result of ever-increasing number of devices connected to the environment (cf. Figure 3.1). The collected data often suffers from various inconsistencies (i.e., anomalies, missing values and redundancies). Redundancy means similar data produced by devices/sensors in a connected environment [57, 58]. Redundancy in data can be temporal and spatial [57]. Temporal redundancy is observed in a single device over a period of time or multiple devices at same time stamp. Spatial redundancy is produced by devices deployed in close vicinity of each other.

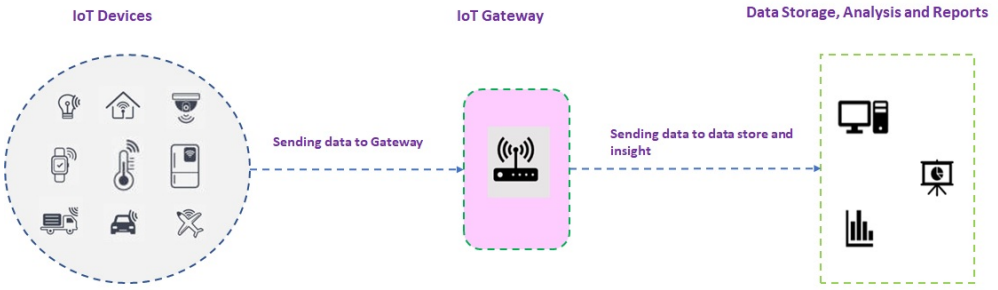


FIGURE 3.1: Data Processing in Connected Environments

We propose here DRMF (Data Redundancy Management for leaF-edge) to identify and remove data redundancies in connected environments at the device level. DRMF considers both static and mobile sensing devices when identifying/detecting redundancies within the generated sensor observations. It provides two algorithms for redundancy detection: the first relies on the temporal feature (specifically designed for static/immobile devices), and the second relies on both temporal and spatial features (specifically designed for mobile devices). The algorithms' parameters are automatically tuned based on historical device data, in order to identify relevant redundancy partitions (e.g., groups of similar-enough data that are considered duplicates). Once redundancies are identified, DRMF proposes a data deduplication module that takes into account the requirements of data consumers, edge data redundancies at the edge and core of the network, the network dynamicity and device mobility, device/network resources (e.g., processing, battery, memory, bandwidth), and personalized ready-to-use data sets for data consumers. Experimental results highlight the performance and accuracy of our solution in detecting and eliminating edge data redundancies.

The remainder of this chapter is organized as follows. Section 3.2 formally defines the terminology used throughout our contribution in order to clarify the terms and focus on the different types of redundancies that we target in this chapter. Then, Section 3.3 presents the DRMF framework, explains its different modules, and zooms in on the proposed algorithms for redundancy detection at the device level. Section 3.4 evaluates the proposal by detailing the experimental protocol, describing the conducted experiments, and analyzing the obtained results. Finally, Section 3.5 recaps this chapter and highlights the potential improvements of DRMF.

3.2 Preliminaries

In this section, we formally define the key concepts of this study. Since DRMF's aim is to detect redundancies at the source (i.e., on devices), it targets the sensed data or observations that the embedded sensors record over time and in different locations. Therefore, it is important to define the structure of the aforementioned data. To do so, we first define temporal stamps in Definition 1 and location stamps in Definition 2 to detail the spatial-temporal attributes of the sensed data. Then, we formally define data items (cf. Definition 3) which represent individual sensor observations or sensed data.

Definition 1 (Temporal Stamp)**.** *A temporal stamp t designates a single discrete temporal value formally defined as a 2-tuple:*

$$t = \langle format, value \rangle \quad where: \tag{3.1}$$

- $format$ *is a string indicating the format of the date-time value of t (e.g., "dd-MM-yyyy hh:mm:ss")*
- $value$ *is the timestamp value (e.g., 10-11-2020 15:34:23 following the sample time format mentioned above)* ■

Definition 2 (Location Stamp)**.** *A location stamp l is s a discrete and instantaneous location value defined as a 2-tuple:*

$$l = \langle format, value \rangle \quad where: \tag{3.2}$$

- $format$ *is the location referential format following which the location stamp value will be represented (e.g., default GPS, or Cartesian, Spherical, Cylindrical)*
- $value = \langle x, y, z \rangle$ *is a discrete and instantaneous value, where x, y, and z designate individual coordinate values (the coordinates can be translated into the referential of choice following the designated format)* ■

Definition 3 (Data Item)**.** *We formally define a data item d as a 5-tuple:*

$$d : \langle a, v, t, l, s \rangle \quad where: \tag{3.3}$$

- *a is the data attribute*
- *v is the data value*
- *t is the creation temporal stamp of d (cf. Definition 1)*
- *l is the creation location stamp of d (cf. Definition 2)*
- *s is the data source that produced/created d* ■

Table 3.1 shows an excerpt of the locally stored data on a particular device. The latter embeds two sensors $S1$, and $S2$ that sense humidity and temperature observations respectively. Each row in the table represents an individual data item having an attribute, data value, temporal stamp, location stamp, and source.

TABLE 3.1: Data Items Example

Attribute	Value	Temporal Stamp		Location Stamp		Source
		Format	Value	Format	Value	
Humidity	98	dd/mm/yyyy hh:mm:ss	10/02/2019 10:00:00	cartesian	$(8, 12, 8)$	S1
Humidity	109	dd/mm/yyyy hh:mm:ss	10/02/2019 10:02:00	cartesian	$(6, 8, 6)$	S1
Humidity	110	dd/mm/yyyy hh:mm:ss	10/02/2019 10:04:00	cartesian	$(2, 4, 8)$	S1
Humidity	111	dd/mm/yyyy hh:mm:ss	10/02/2019 10:06:00	cartesian	$(4, 6, 4)$	S1
Temperature	22	dd/mm/yyyy hh:mm:ss	10/02/2019 10:08:00	cartesian	$(6, 4, 8)$	S2

Data redundancies in connected environments can be caused by various reasons (e.g., overlapping sensor coverage, sensing when no changes occur in the environment). Regardless of the reasons, redundancies are not normally a punctual phenomena. They often spread over continuous periods of time and/or spatially defined zones/areas. Since DRMF aims to detect data redundancies on static and mobile devices, it is important to track the temporal coverage of redundancies when it comes to static devices, and the spatial-temporal coverage of redundancies for mobile sensors. To do so, we formally define in the following temporal and spatial coverage (cf. Definitions 4, 5 respectively).

Definition 4 (Temporal Coverage)**.** *A temporal coverage $coverage_t$ is a time interval consisting of an ordered collection of temporal stamps enclosed within a start and an end stamp, describing the temporal coverage of a sensor observation (e.g., video feed) or a group of observations (e.g., scalar measurements, images). Formally, it is defined as a 2-tuple:*

$$coverage_t = \langle \delta_t, g_t \rangle \quad where: \tag{3.4}$$

- $\delta_t = [t_s, t_e]$ *is a temporal interval where:*
  - $t_s < t_e$ *is the start temporal stamp*
  - t_e *is the end temporal stamp*
- g_t *is a temporal granularity or unit of the temporal coverage (e.g., millisecond, second, minute, etc.)* ■

Definition 5 (Location Coverage)**.** *A location coverage $coverage_l$ is the set of spatial stamps designating the surface coverage in which a sensor observation is created (e.g., area in which a video stream or a bunch of mobile measurements are recorded). Formally, it is defined as a 2-tuple:*

$$coverage_l = (\delta_l, g_l) \quad where: \tag{3.5}$$

- $\delta_l = \langle shape, L \rangle$ *defines the area of the location coverage where:*
  - $L = \bigcup_{i=0}^{n} l_i \forall i \in \mathbb{N}$ *is a set of location stamps*
  - $shape$ *is a mathematical abstraction used to describe the location coverage, as a continuous coverage area (e.g., rectangle, circle), or non-continuous coverage area (e.g., disk, path, polygon, random)*
- g_l *is the location granularity or unit of the location coverage (e.g., millimeter, centimeter, meter).*

Remark. *The shape of a location coverage depends on the sensors and the environment where they are deployed. For instance, the shape could be lines (for mobile sensor tracking), continuous rectangles (e.g., in an office), or non-continuous disks or random shapes (e.g., in a forest excluding lakes).* ■

Based on the aforementioned definitions, we finally define temporal and spatial-temporal redundancies (cf. Definitions 6, 7 respectively). DRMF targets these two types of redundancies for static and mobile devices. Since static devices are immobile, we only monitor the spread of the redundancy over time. This is not the case for mobile devices since they can move around in the environment and change locations over time. Therefore, we monitor the spatial-temporal coverage for redundancies on mobile devices.

Definition 6 (Temporal Redundancy)**.** *A temporal redundancy tr is defined as a 2-tuple:*

$$tr : (coverage_t, D) \ \ where: \tag{3.6}$$

- $coverage_t$ *is the temporal coverage during which the data is temporally redundant (cf. Definition 4)*
- $D = \bigcup_{j=0}^{z} d_j$ *is a cluster of redundant data items where:*
  - $\forall\ d_j \in D,\ d_j.t \in coverage_t.\delta_t$
  - $\forall\ d_{j1}, d_{j2} \in D,\ d_{j1}.a = d_{j2}.a$
  - $\forall\ k \in \mathbb{N}^+,\ d_k.v = d_{centroid}.v \pm \delta_v$ *where:*
    - $d_{centroid}.v$ *is the centroid value of all data items in* D
    - δ_v *is an acceptable deviation threshold*

Remark. *The threshold δ_v is calculated based on the data distribution within the redundant data set D.* ■

Definition 7 (Spatial-Temporal Redundancy)**.** *A spatial-temporal redundancy str is defined as a 3-tuple:*

$$str : (coverage_t, coverage_l, D) \ \ where: \tag{3.7}$$

- $coverage_t$ *is the temporal coverage (cf. Definition 4)*
- $coverage_l$ *is the location coverage (cf. Definition 5)*
- $D = \bigcup_{j=0}^{z} d_j$ *is a cluster of redundant data where:*
  - $\forall\ d_j \in D,\ d_j.t \in coverage_t.\delta_t$
  - $\forall\ d_j \in D,\ d_j.l \in coverage_l.\delta_l$
  - $\forall\ d_{j1}, d_{j2} \in D,\ d_{j1}.a = d_{j2}.a$
  - $\forall\ k \in \mathbb{N}^+,\ d_k.v = d_{centroid}.v \pm \delta_v$ *where:*
    - $d_{centroid}.v$ *is the centroid value of all data items in* D
    - δ_v *is an acceptable deviation threshold* ■

3.3 DRMF: Data Redundancy Management for leaF-edge

We introduce here our proposal DRMF. In this study, we show how DRMF handles sensor data redundancies at the edge device level, considering both static and mobile devices, in order to eliminate redundancies from the source before reaching the core of the network. The DRMF overall architecture is depicted in Figure 3.2. In a typical CE, the sensor observations are temporarily stored in the device's memory, before the data is eventually transmitted to a permanent storage repository or another device. In DRMF, we propose to detect and handle redundancies prior to data storage or transmission. Therefore, we represent the locally stored data (i.e., the set of data items as defined in Definition 3) on devices via the *Edge Device Data* layer. Then, the *Device Level Redundancy Management* layer takes as input the stored data/observations and detects either temporal or spatial-temporal redundancies within it. The aforementioned process consists of three steps represented by the following modules: (i) datatype filtering; and (ii) redundancy detection; and (iii) redundancy detection tuning. In the following, we describe each module separately and explain how they interact together.

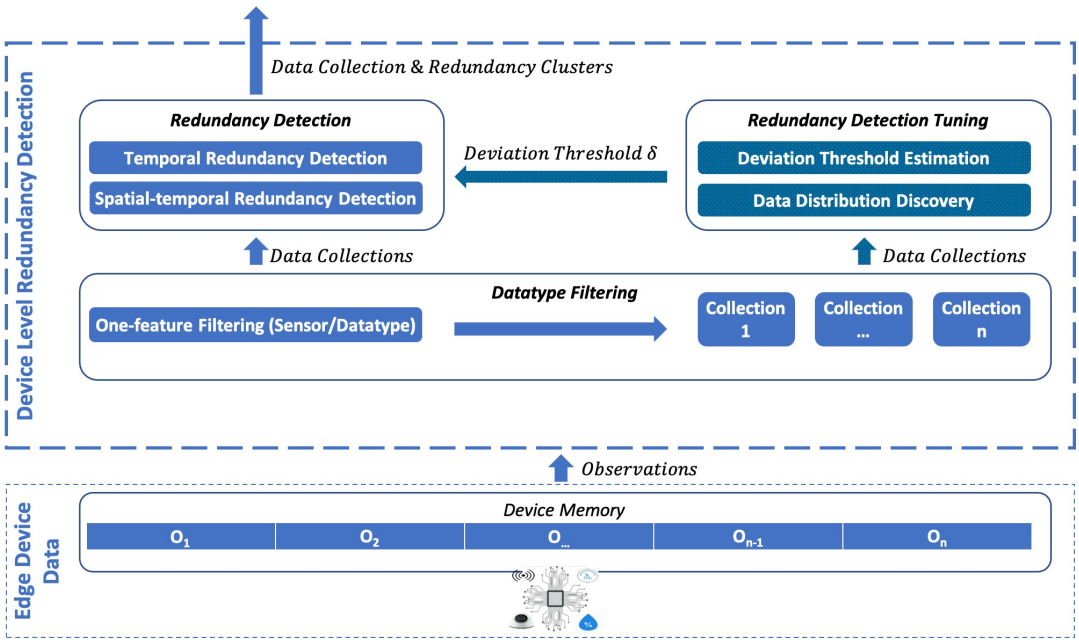


FIGURE 3.2: Processing of DRMF

3.3.1 Data Type Filtering

This module receives sensed data from devices/sensors. A device can have one or more sensors each sensing a specific attribute (e.g., temperature, humidity, CO_2). First, it filters data items based on their attributes since it does not make sense to check fore redundancies on a heterogeneous set of attributes (e.g., humidity and temperature). As a result, separate data collections are generated for similar data items (i.e., data items having the same attribute). In the next step, we will look for redundancies within each data collection. We should also note that we consider that a sensor can only sense one attribute. Therefore, the filtering can also be done on the source instead of the attribute itself.

To illustrate this process, consider the example provided in Table 3.1. It shows two different attributes (humidity and temperature) locally stored together in the

device's memory. Therefore in order to detect redundancies, one must start by filtering the data into collections having the same attributes (or datatypes). To illustrate the datatype filtering process, the data shown in Table 3.1 produces two distinct data collections: the first for humidity data containing the first four tuples (cf. Table 3.2); and the second for temperature data containing the last tuple (cf. Table 3.3).

TABLE 3.2: Humidity Data Collection

Attribute	**Value**	**Temporal Stamp**		**Location Stamp**		**Source**
		Format	**Value**	**Format**	**Value**	
Humidity	98	dd/mm/yyyy hh:mm:ss	10/02/2019 10:00:00	cartesian	$(8, 12, 8)$	S1
Humidity	109	dd/mm/yyyy hh:mm:ss	10/02/2019 10:02:00	cartesian	$(6, 8, 6)$	S1
Humidity	110	dd/mm/yyyy hh:mm:ss	10/02/2019 10:04:00	cartesian	$(2, 4, 8)$	S1
Humidity	111	dd/mm/yyyy hh:mm:ss	10/02/2019 10:06:00	cartesian	$(4, 6, 4)$	S1

TABLE 3.3: Temperature Data Collection

Attribute	**Value**	**Temporal Stamp**		**Location Stamp**		**Source**
		Format	**Value**	**Format**	**Value**	
Temperature	22	dd/mm/yyyy hh:mm:ss	10/02/2019 10:08:00	cartesian	$(6, 4, 8)$	S2

3.3.2 Redundancy Detection

Once the data filtering is done, the generated data collections are sent to the redundancy detection module. Based on the type of the device (i.e., static or mobile) one of the following redundancy detection algorithms is triggered: (i) temporal redundancy detection; or (ii) spatial-temporal redundancy detection. Both algorithms cluster the data based on the deviation of the data item values, while also considering the temporal, or spatial-temporal coverage (or spread) of the clusters (i.e., sets of redundant data). We propose to detect redundancies using an unsupervised cluster-based approach for two main reasons: (i) to avoid applying supervised learning which requires training time and computation power on the edge where resources are often limited; and (ii) since training data for supervised learning algorithms might not be available at the device level. In the following, we detail each algorithm separately.

Redundancy Detection Algorithms

Algorithm 1 (temporal redundancy detection) groups the data into clusters of temporally redundant data. It takes a data collection C as input, and produces a set TR of temporal redundancies (clusters) as output. First, the algorithm sorts all data items in the input collection by ascending time. Then, for each data item, the algorithm checks if a cluster already exists. If not, a new cluster is created with the current data item added as its centroid (lines 3-6). However, if a cluster already exists, the algorithm checks if the current data item belongs to the aforementioned cluster. This is done by measuring the distance between the data item and the cluster centroid

values and comparing it to a deviation threshold δ_v (line 8). If the current data item belongs to the cluster, a new centroid is computed and the algorithm checks the next value in the collection (lines 9-10). This step is repeated until the algorithm finds a value that does not belong to the cluster. In this case, the temporal coverage of the cluster is calculated (lines 12-13), the cluster (i.e., temporal redundancy) is added to the output list (line 14), and the variable cluster content (D) is reset (line 15) in order to generate a new cluster and look for other redundancies.

Algorithm 1: Temporal Redundancy Detection

```
   Input      : C  // C is a data item collection (i.e., a set of data items)
   Output     : TR  // TR is a set of temporal redundancies found within C
   Parameters : δ_v, g_t  // δ_v is a value threshold; g_t is a temporal unit
   Local Variables: SC, cov_t, centroid, D, min_t, max_t, δ_t
          /* SC is the temporally sorted collection; cov_t is a temporal coverage;
    centroid is the centroid of a set of values; D is a cluster of data items;
    min_t is the oldest timestamp in a set; max_t is the most recent timestamp in a
    set; δ_t is a temporal interval */
   // Begin algorithm
 1 Initialize TR ← ∅
 2 SC ← sort_t(C)                          // Sort data items by ascending timestamps
 3 foreach data item d_i ∈ SC do
 4     if (∄ cluster of redundant data D) then
 5         Create new cluster D
 6         Initialize centroid ← d_i.v
 7     else
 8         if (Absolute difference |d_i.v − centroid| ≤ δ_v) then
 9             Add data item to cluster D ← d_i
10             Update centroid ← Avg(all d_i.v ∈ D)
11         else
12             Identify temporal interval δ_t ← [min_t, max_t] of D
13             Compute temporal coverage w.r.t. time unit cov_t ← (g_t, δ_t)
14             Add new temporal redundancy TR ← (cov_t, D)
15             Flush out cluster D
16         end
17     end
18 end
19 Return TR
```

Similarly, Algorithm 2 (i.e., spatial-temporal redundancy detection) takes a data collection as input in order to generate a set of clusters as output, where each cluster represents a spatial-temporal redundancy. The clustering principles are the same in both algorithms. However, the spatial-temporal redundancy checker calculates the spatial coverage for each redundancy (i.e., cluster) in addition to the temporal coverage. This entails keeping track of data location stamps in each cluster (line 11) and calculating the characteristics of the coverage area (lines 13, 15, and 17). Note that both clustering algorithms calculate the temporal and spatial-temporal coverage of each cluster respectively, in order to keep track of the temporal and spatial spread of each redundancy.

To illustrate the temporal redundancy detection process, consider the humidity

Algorithm 2: Spatial-Temporal Redundancy Detection

Input : C // C is a data item collection (i.e., a set of data items)
Output : STR // STR is a set of spatio-temporal redundancies found within C
Parameters : δ_v, g_t, g_l // δ_v is a value threshold; g_t is a temporal unit, g_l is a location unit
Local Variables: $SC, cov_t, cov_l, centroid, D, min_t, max_t, L, \delta_t, \delta_l, shape$

/* SC is the temporally sorted collection; cov_t is a temporal coverage; cov_l is a location coverage; $centroid$ is the centroid of a set of values; D is a cluster of data items; min_t is the oldest timestamp in a set; max_t is the most recent timestamp in a set; L is a set of locations; δ_t is a temporal interval; : $delta_l$ is a location area; $shape$ is a geometrical shape */

// Begin algorithm

1 Initialize $STR \leftarrow \varnothing$
2 $SC \leftarrow sort_t(C)$ // Sort data items by ascending timestamps
3 **foreach** *data item* $d_i \in SC$ **do**
4 **if** *(* $\nexists$ *cluster of redundant data* D*)* **then**
5 *Create new cluster* D
6 Initialize $centroid \leftarrow d_i.v$
7 **else**
8 **if** *(Absolute difference* $|d_i.v - centroid| \leq \delta_v$*)* **then**
9 *Add data item to cluster* $D \leftarrow d_i$
10 $Update\ centroid \leftarrow Avg(all\ d_i.v \in D)$
11 *Add data item location to* $L \leftarrow d_i.l$
12 **else**
13 $Identify\ shape \leftarrow getShape(L)$
14 $Identify\ temporal\ interval\ \delta_t \leftarrow [min_t, max_t]\ of\ D$
15 $Identify\ location\ area\ \delta_l \leftarrow (shape, L)\ of\ D$
16 $Compute\ temporal\ coverage\ w.r.t.\ time\ unit\ cov_t \leftarrow (g_t, \delta_t)$
17 $Compute\ location\ coverage\ w.r.t.\ location\ unit\ cov_l \leftarrow (g_l, \delta_l)$
18 $Add\ new\ spatio-temporal\ redundancy\ STR \leftarrow (cov_t, cov_l, D)$
19 *Flush out cluster* D
20 **end**
21 **end**
22 **end**
23 Return STR

data collection presented in Table 3.2. If we apply the temporal redundancy detection algorithm with a deviation threshold $\delta_v = 3$, we detect one temporal redundancy (containing values 109, 110, and 111) and spanning over a temporal coverage of 4 minutes (from 10/02/2019 10:02:00 till 10/02/2019 10:06:00).

3.3.3 Redundancy Detection Tuning

The aforementioned algorithms cluster redundant data within collections based on the value deviation threshold δ_v which can be adjusted per device as a system parameter. However, one does not always know how to choose the optimal value for δ_v. This is important since the set value will affect the accuracy of the process. Therefore, we propose here the redundancy detection tuning module (cf. Figure 3.2) to automatically set, and re-adjust if necessary, the deviation threshold (algorithms' parameter) based on historical data. To do so, the filtered data collections are sent to the data distribution discovery module that identifies the distribution of the sensor observation values using tests such as Chi-Squared and Kolmogrov-Smirnov [1]. Once a distribution is identified (e.g., Normal, Gamma, Beta, Exponential, Weibull, Bernoulli)[2], we estimate the deviation threshold δ_v using one of the following techniques: (i) MAD: Mean Absolute Deviation; (ii) Z-score; and (iii) IQR: Interquartile Range. The estimated threshold is used for a specific type of collection (e.g., temperature) when identifying temporal or spatial-temporal redundancies. In order to avoid repeating this process at each run, we keep using the same threshold until the accuracy of the deduplication drops below a specific acceptable level. Only then, the redundancy detection tuning process is triggered again and the threshold is re-estimated and adjusted accordingly.

3.4 DRMF Evaluation

In this section, we evaluate the performance and accuracy of the DRMF redundancy detection algorithms. We start first with a theoretical complexity analysis. Then, we detail our experimental protocol, tests, and results.

3.4.1 Complexity Evaluation

This section discusses complexity analysis of the redundancy detection process. More precisely, we analyze algorithms 1 and 2 by taking into consideration the temporal and spatial complexities. This analysis of resource consumption (both time and space) takes into account the impact of the input data size and the chosen deviation threshold. This provides a theoretical analysis about run-time and memory usage behaviour based on the input size and the chosen deviation threshold (algorithm parameters). Specifically, we evaluate the Big O ($\mathcal{O}$) and Big Omega (Ω) notations to estimate the worst and best cases respectively:

- Big($\mathcal{O}$): If we describe an algorithm by a function $f(n)$, then Big($\mathcal{O}$) of $f(n)$ is a function $g(n)$ that bounds it (i.e., after a specific value, $g(n)$ would always be greater than $f(n)$). We find the following common notations of the Big($\mathcal{O}$): (i) $\mathcal{O}(1)$ is used to describe that an algorithm will always execute in the constant time (or space) irrespective of the size of the input; (ii) $\mathcal{O}(n)$ specifies growth rate of an algorithm is linear with respect to resources and is directly

[1] https://towardsdatascience.com/identify-your-datas-distribution-d76062fc0802
[2] https://www.kdnuggets.com/2020/06/overview-data-distributions.html

proportional to size of the input; (iii) $\mathcal{O}(n^2)$ describes growth rate of an algorithm is directly proportional to the square of the size of the input. This growth rate is found in algorithms that involve nested iterations (e.g., deeper nested iterations result in $\mathcal{O}(n^3)$, $\mathcal{O}(n^4)$); and (iv) $\mathcal{O}(2^n)$ represents growth rate of recursive algorithms for instance.

- Big(Ω): We use this notation in complexity analysis for denoting best case scenario of an algorithm (i.e., the minimum amount of required resources). If we describe an algorithm by a function $f(n)$, then Big(Ω) of $f(n)$ is a function $g(n)$ that bounds the lower end of $f(n)$ (i.e., after a specific value $f(n)$ would always be more than $g(n)$).

Algorithm 1 Evaluation

Time Complexity The temporal redundancy detection algorithm receives as input a filtered collection of data. This algorithm finds clusters of redundant data items within specific temporal coverages/spreads. The input size of data collection is $N = \|C.len\|$. The algorithm first initializes a set TR which would hold clusters of temporally redundant data. Then the function $sort_t$(C) sorts the data collection in ascending order with respect to time. The algorithm then executes the foreach loop that goes through the sorted collection of data items and generates clusters of temporally redundant data based on difference between deviation threshold and data items' values. Besides initializing variables and values, it updates these values when it finds new members of redundant cluster. The algorithm also uses two nested if/else statements: (i) to check whether a cluster of redundant data items exists or not (line 4); and (ii) to find out whether the current data item belongs to existing cluster or not (line 9). All statements inside the loop are executed N times.

In the worst case, Big ($\mathcal{O}$) of Algorithm 1 is $\mathcal{O}(N)$. However, in the best case when input size of data collection (i.e., $N = 1$ when a data collection C has only 1 data item), the Big (Ω) is $\Omega(1)$. This shows that in theory, the complexity of Algorithm 1 with respect to time is linear and is directly proportional to input size of data.

Space Complexity The space complexity of Algorithm 1 is proportional to the input size. It has variables for storing sum and centroid values. The values of these variables change with each iteration. Algorithm 1 (line 5) creates a cluster of redundant data if it does not already exist. This cluster is stored in the variable D which takes some memory resources. Besides this, the algorithm also stores clusters of redundant data in TR (line 18), which is a combination of all temporally redundant clusters. TR also occupies space in memory.

In worst case, space complexity of Algorithm 1 is $\mathcal{O}(N)$. However, in the best case scenario when input size of data collection (i.e., $N = 1$) the space complexity would be $\Omega(1)$. This shows that in theory, the complexity of Algorithm 1 with respect to space is linear and is directly proportional to input size of data.

Algorithm 2 Evaluation

Time Complexity. The spatial-temporal redundancy detection algorithm receives as input a filtered collection of data. This algorithm finds clusters of redundant data

items within spatial-temporal coverages/spreads. The input size of data collection is $N = \|C.len\|$. The algorithm first initializes a set STR which would hold clusters of spatial-temporally redundant data. First, the function $sort_t$(C) sorts the data collection in ascending order with respect to time. Then, the algorithm runs the foreach loop that goes through the sorted collection of data items and generates clusters of spatial-temporally redundant data based on difference between deviation threshold and data items' values. Besides initializing variables and values, it updates these values as it finds new members of redundancy clusters. Algorithm 2 also uses two nested if/else statements: (i) to check whether a cluster of redundant data items exists or not (line 4); and (ii) to find out whether the current data item belongs to existing cluster or not (line 9). All statements inside the loop are executed N times.

In worst case, the Big ($\mathcal{O}$) of Algorithm 2 is $\mathcal{O}(N)$. However, in best case when the input size of a data collection C is equal to one ($N = 1$), the best case Big (Ω) is $\Omega(1)$. This indicates that in theory, the complexity of Algorithm 2 with respect to time is linear and is directly proportional to input size of data.

Space Complexity. The space complexity of Algorithm 2 is proportional to the input size. Algorithm 2 has variables for storing sum and centroid values. The values of these variables change with each iteration. Algorithm 2 generates a cluster of redundant data if it does not already exist (line 5). This cluster is stored in D which takes some memory resources. Besides this, it also stores clusters of redundant data in STR (line 20), which is a combination of all spatial-temporally redundant clusters. STR also occupies space in memory.

In worst case, space complexity of Algorithm 2 is $\mathcal{O}(N)$. However, in best case when the input size of a data collection C is one ($N = 1$), the space complexity would be $\Omega(1)$. This shows that in theory, the complexity of Algorithm 2 with respect to space is linear and is directly proportional to input size of data.

Theoretical Complexity Comparison

Table 3.4 represents a complexity comparison between several existing approaches. If we analyse 3.4, we find that the time complexity of the proposed approach is better than most of the existing techniques. There are a few approaches [31, 54, 55] who have almost the same time complexity but they process data at the cluster head or aggregator level rather than the device level. Another approach [50] represents almost the same time complexity but it clusters the sensor nodes rather than data.

3.4.2 Experiments and Results

We present here the implementation and evaluation of our proposed approach. We provide details of experiments and results of performance evaluation for temporal and spatial-temporal data redundancy detection at device level. We conducted performance tests with an Intel CORE i5 1.8 GHz 8th generation processor and 16 GB of RAM. We developed a prototype in Python 3.8 using the PyCharm IDE. The aim of the experiments was to test the performance of proposed DRMF. We present first the experimental protocol in order to evaluate both algorithms. We detail the objectives of the experimentation, the evaluation metrics, and the experiments. Then, we present performance results for Algorithms 1 and 2.

TABLE 3.4: Temporal Complexity Comparison

Approach	**Time Complexity**
Alam M. K. et al. [52]	$\mathcal{O}(N \times K)$
Baba A. et al. [28]	$\mathcal{O}(N^2)$
Chowdhury S. et al. [50]	$\mathcal{O}(U + E)$
Idrees A. et al. [34]	$\mathcal{O}(t \times p \times k)$
Ismael W et al. [31]	$\mathcal{O}(N + M)$
Li S. et al. [35]	$\mathcal{O}(N^3)$
Qurabat A. et al. [9]	$\mathcal{O}(N^3)$
Karim F. et al.[56]	$\mathcal{O}(N^2)$
Yemeni Z. et al. [47]	$\mathcal{O}(N^2)$
Ismael W. et al. [48]	$\mathcal{O}(N^2)$
DRMF [20]	$\mathcal{O}(N)$

Experimental Protocol

Experimentation Objectives The objectives of the experimentation are two-fold: (i) highlighting the proposal's ability to detect redundancies accurately - this requires evaluating the accuracy of the clustering algorithms when detecting redundancies and comparing them with existing works; and (ii) highlighting the feasibility of implementing the proposed approach at device level - this requires evaluating the performance of our proposal in order to show that the costs are acceptable at the network edge (where resources are often limited).

Dataset We used the Intel Berkeley Lab dataset[3] for performance evaluation. The dataset was obtained by 54 Mica2Dot sensors. The dataset provides data for weather such as temperature, humidity, light and voltage. It also provides timestamps at which the data was acquired. In our experiments, we used the data of one *humidity* sensor from 28^{th} February 2004 to 31^{st} March 2004. The dataset includes a total of 489212 records. We used 38656 records for one humidity sensor after removing negative values.

Evaluation Metrics To evaluate the performance of DRMF, we measured the run-time, CPU consumption, and required RAM size during a set of experiments. Moreover, to evaluate the accuracy of the redundancy detection process we measured the deduplication accuracy and ratio. Finally, to evaluate the redundancy detection tuning module of DRMF (cf. Figure 3.2), we evaluated the threshold estimation process.

Performance Evaluation

Proposed Experiments We ran the following experiments on a Dell machine with Windows 10, having a Core i5 8^{th} Generation 1.8 GHZ processor, and 16 GB of RAM. We evaluate the performance of both redundancy detection algorithms and the auto-clean mode for redundancy removal (we left the evaluation of the consumer-centric deduplication for a separate work) by measuring the run-time, CPU consumption, and RAM size.

- Experiment 1: *Input Data Size Impact*. In this test, we gradually increase the input data size in order to assess its impact on performance.

[3] http://db.csail.mit.edu/labdata/labdata.html

- Experiment 2: *Deviation Threshold Impact.* In this test, we gradually increase the deviation threshold to generate clusters with various sizes and spreads to assess its impact on performance.

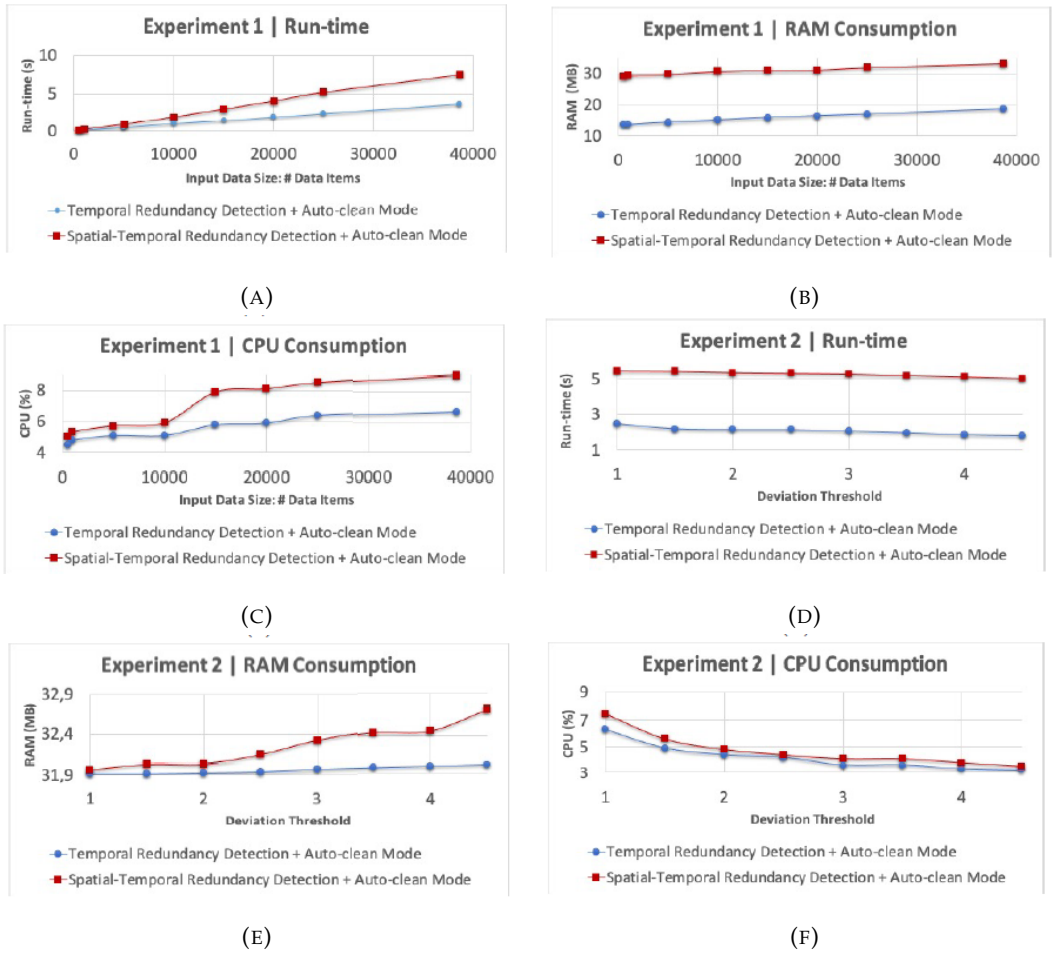


FIGURE 3.3: Performance Results: Experiments 1 and 2

Discussion. Figures 3.3a, 3.3b, and 3.3c show the results of experiment 1. Increasing the number of input data items from 0 to 38656 had a visible impact on performance. The required time, CPU, and RAM for identifying and removing redundancies increase in a quasi-linear way. However, in the worst case scenario (i.e., 38656 values) the required time does not exceed 8 seconds and the required RAM/CPU does not surpass 34 MB and 9% respectively. Figures 3.3d, 3.3e, and 3.3f show the results of experiment 2. Increasing the deviation threshold generates fewer but bigger clusters of redundant data. This is reflected in the results: the bigger the threshold, the less time and CPU are required since less clusters are generated. RAM consumption slightly fluctuates between 31.9 and 32.9 MB, yet the difference is not significant since every data item eventually belongs to one cluster regardless of the number of generated clusters. Finally, for both tests, Algorithm 2 requires more resources since it considers both temporal and spatial dimensions in contrast with Algorithm 1 which only consider the temporal dimension.

Accuracy Evaluation

Proposed Experiments We also evaluate the accuracy of our proposal by measuring data reduction accuracy (using the Jaccard Similarity Index), and the data reduction ratio. Each experiment is done twice (with temperature and humidity values). For the two cases, we identify redundancies using both algorithms and clean duplicates by summarizing every redundancy cluster to one representative data item (e.g., the mean) for the experimentation purposes. We present here the temporal redundancy identification and cleaning results (a more detailed discussion about cleaning is provided in Chapter 6).

- Experiment 3: *Input Data Size Impact.* In this test, we gradually increase the input data size to assess its impact on accuracy.
- Experiment 4: *Deviation Threshold Impact.* In this test, we gradually increase the deviation threshold to assess its impact on accuracy.
- Experiment 5: *Threshold Estimation Impact.* In this test, we change the threshold estimation technique (i.e., Z-score, MAD, IQR) and measure the data reduction accuracy and ratio using the estimated thresholds.

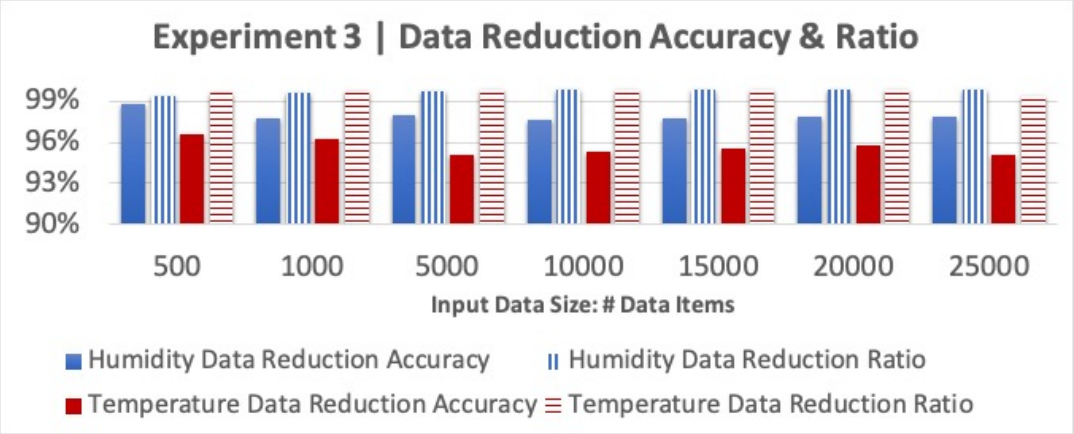


FIGURE 3.4: Experiment 3 Results

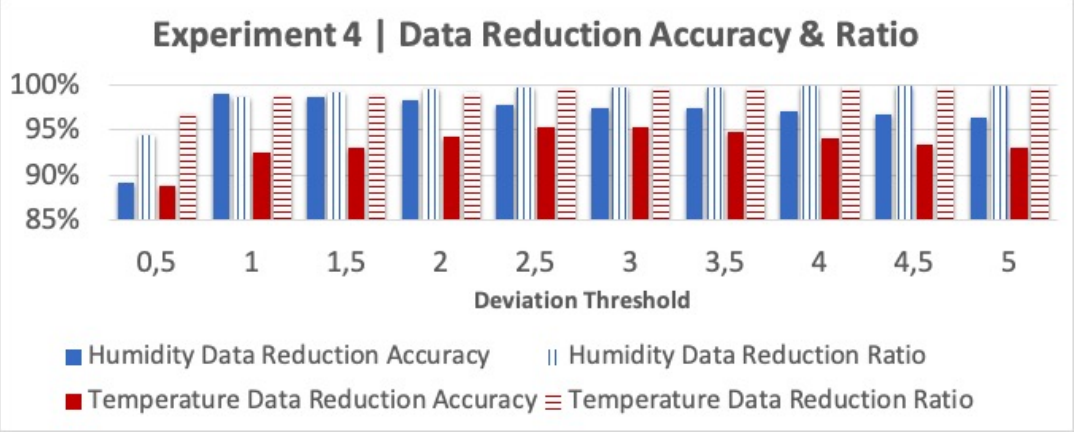


FIGURE 3.5: Experiment 4 Results

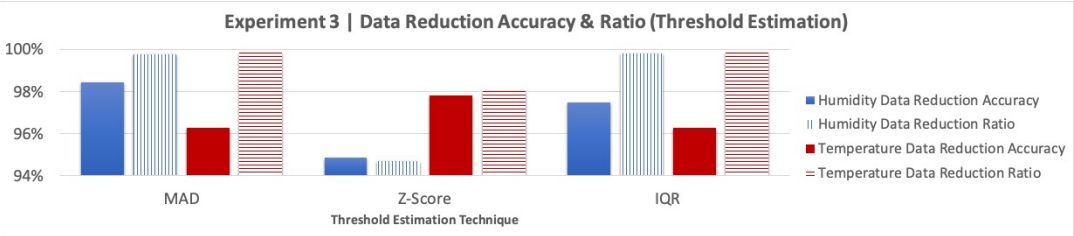


FIGURE 3.6: Experiment 5 Results

Discussion. Figure 3.4 shows the results of experiment 3. The data reduction accuracy (with a fixed deviation threshold of 2.5) varies between 97.63% and 98.78% for humidity data. Moreover, accuracy varies between 95.03% and 96.54% for temperature. Figure 3.5 shows the results of experiment 4. Increasing the deviation threshold affects the clustering of redundancies. The optimal deviation threshold in the 0.5 to 5 range is 1 for humidity (accuracy of 99.10%) and 3 for temperature (accuracy of 95.43%). In addition, both experiments achieve high deduplication ratios (94.48% to 99.90%). Finally, Figure 3.6 shows that results of experiment 5 which highlights the importance of choosing the adequate technique based on the data distribution and historical patterns (i.e., the technique should not be randomly assigned/configured). The discovered distribution for both temperature and humidity collections is Weibull. This explains why Z-Score provided the worst result since it is based on the mean and standard deviation (i.e., more suited for normal distributions).

3.5 Recap

In this chapter, we address the need for detecting redundancies at the source (i.e., at the device level) while considering both static and mobile devices in connected environments (cf. Need 1 - Challenge 1 in Chapter 1). After reviewing the literature in Chapter 2, we discuss here the data redundancy detection mechanism of our proposal DRMF. We define the key terminology that we used in this approach before presenting its data redundancy detection algorithms. Furthermore, we presented a detailed experimentation of our proposal and discussed the obtained results. The key advantage of our proposal is its capability of delivering accurate results without requiring excessive computational complexity. However, DRMF is a clustering-based redundancy detection proposal. It groups redundant data based on a specified crisp threshold that bounds the clusters and affects the decision making about the inclusion or not of a data item in a particular cluster. This constitutes a limitation in particular use cases. If the threshold is not very precise, this could lead to either not detecting some redundant data or falsely flagging a data item as redundant. In the next chapter, we discuss an improvement of DRMF that utilizes fuzzy reasoning to address the aforementioned issue.

Chapter 4

FREDD: Fuzzy Redundancy Elimination for Data Deduplication

In the previous chapter, we addressed data redundancy at the CE edge level by providing clustering algorithms capable of detecting data redundancies locally on edge devices. The proposal clusters redundant data values into redundancy clusters by comparing their deviation to a crisp threshold (cluster boundary). Although this constitutes an efficient way of detecting redundancies at the edge, it is not optimal since data values that slightly deviate from the crisp cluster boundaries might be wrongly assigned to a cluster.

In this chapter, we address this limitation by using fuzzy logic to overcome the shortcomings of relying on crisp cluster boundaries. To do so, we propose FREDD (Fuzzy Redundancy Elimination for Data Deduplication). We provide first a background on fuzzy logic. Then, we describe the FREDD processing steps in order to show how we use fuzzy logic to improve data redundancy detection on static and mobile edge devices in a CE.

Thereafter, we present an extensive evaluation of FREDD by describing the experimental protocol, the conducted experiments, and the obtained results. Finally, we discuss how we can extend FREDD to consider redundancies at the core of the network (i.e., at the sink level).

Sylvana Yakhni, Joe Tekli, Elio Mansour, Faisal Shahzad, and Richard Chbeir. Fuzzy Redundancy Elimination for Data Deduplication in Connected Environments. Submitted to the Journal of Parallel and Distributed Computing.

4.1 Introduction

As described in Chapter 2, most existing approaches for data handling and preprocessing in CEs share various limitations. Existing solutions mostly disregard the dynamicity and constraints of the network, considering static devices only (e.g., stationary surveillance cameras, pollution sensors), e.g., [31, 35, 53], overlooking mobile devices (e.g., phones, tablets). Also, they provide the user with minimum-to-no control over the deduplication process (which data need to be duplicated and which data should be kept intact) hence overlooking the user's requirement and application needs in defining redundancy, e.g., [20, 53].

In this chapter, we propose: FREDD, a new approach for Fuzzy Redundancy Elimination for Data Deduplication in a connected environment. FREDD uses simple natural language rules to represent expert knowledge and user preferences regarding data duplication boundaries. It then applies pattern codes and fuzzy reasoning to detect duplicates at the edge level of the network. This reduces the time required to hard-code the deduplication process, while adapting to the user's needs for different data sources and applications. Moreover, FREDD is adapted for both static and mobile devices. Experiments on a real-world dataset highlight FREDD's potential and improvement compared with existing solutions.

In the following, Section 4.2 provides a preliminary on fuzzy logic in order to cover the basics of using fuzzy reasoning. Then, Section 4.3 describes the FREDD framework and details each of its modules. We also illustrate the processing steps with a running example that shows how fuzzy reasoning contributes the redundancy detection at the device level. Section 4.4 provides a complexity evaluation of FREDD as well as an extensive experimentation that assesses the accuracy and performance of this proposal. Finally, Section 4.5 summarizes the chapter and discusses the extension of FREDD to provide redundancy detection at the sink level in a connected environment.

4.2 Fuzzy Logic Preliminaries

Fuzzy logic is a multi-valued logic that allows the definition and usage of intermediate values between conventional evaluations like true/false, yes/no, duplicate/not duplicate, etc. It is a paradigm for processing data by using partial set membership, where an element can be part of one set and its compliment albeit with varying membership degrees (e.g., 70% true and 30% false). It incorporates a condition-action rule-based IF X AND Y THEN Z approach rather than attempting to model a system mathematically [30]. The model and its fuzzy membership functions are defined empirically, and rely on the designer's experience and understanding of the system and its environment [59]. For example, rather than dealing with data values in terms of humidity = 95 µg/m3 and temperature = 18.2 ^{0}C, expressions like IF Low(humidity_value1) AND VeryHigh(humidity_value2) THEN NotDuplicate(status) are used. While they seem imprecise, yet such expressions can be very descriptive and provide a necessary level of abstraction on top of the crisp data values, allowing to guide the decision making process. A typical fuzzy logic agent consists of 5 main components [30, 60]: i) fuzzification, ii) condition-action rules, iii) inference, iv) aggregation, and v) defuzzification. Fuzzification consists in transforming input crisp values (received from sensors) into fuzzy membership

scores associated with a set of linguistic variables (e.g., low humidity, high temperature) defined by the system designer (e.g., humidity = 95 µg/m3 is transformed into 25% low and 75% medium humidity). Condition-action rules are defined as Boolean logic (IF-THEN) expressions that reflect the common sense logic applied by a human expert to guide the decision making process. Inference consists in applying a set of designate condition-action rules on the fuzzified data in order to produce fuzzy outputs. Multiple rules can produce different outputs, and need to be aggregated in order to produce one single fuzzy output function. The fuzzy output function is consequently defuzzified in order to produce crisp values as the final output of the agent.

We adopt the fuzzy logic paradigm in order to automate the redundancy detection process in a connected environment, while handling the different needs and challenges mentioned in our motivation scenario (cf. Chapter 1). We detail next the FREDD framework.

4.3 FREDD Framework

To address the above mentioned challenges, we introduce FREDD: a new framework for Fuzzy Redundancy Elimination for Data Deduplication in a connected environment. FREDD detects data duplicates at the edge level, considers data redundancies from static and mobile devices (cf. Challenge 1), and combines simple natural language rules with a fuzzy inference mechanism designed to adapt the deduplication process following the user's needs (cf. Challenge 3). FREDD's core architecture for edge-level deduplication is depicted in Figure 4.1. It consists of six main modules: (i) **sensor data representation** which defines the spatial and temporal representations of data measurements/items; (ii) **attribute separation** which separates the input data into attribute-based data collections; (iii) **pattern code generation** which associates data items with pattern codes based on user-defined lookup tables; (iv) **duplicate candidate filtering** which determines whether data items are candidates for fuzzy duplication; and (v) **fuzzy redundancy detection** which identifies duplicate data items using fuzzy reasoning based on user-defined condition-action rules. We describe each of the latter modules for edge-level deduplication in the following sub-sections. We detail the redundancy removal process extensively in Chapter 6.

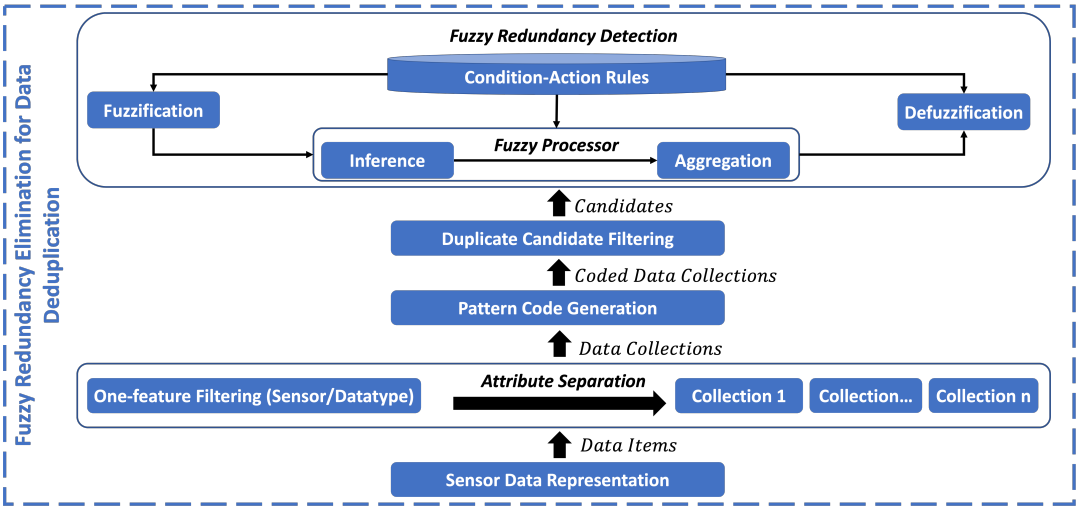


FIGURE 4.1: FREDD's Architecture

4.3.1 Sensor Data Representation

Connected environments contain diverse devices each embedding one or more sensors that provide data from the real world. Static devices are immobile; therefore, the data generated by such devices can be redundant temporally. However, mobile devices produce data while moving around the environment, which potentially generates spatial-temporal redundancies. We maintain the same definitions of data items, temporal/location stamps/coverage, and temporal/spatial-temporal redundancies introduced in Chapter 3 - Section 3.2. To illustrate the FREDD redundancy detection process consider the following set of data items listed in Table 4.1. This example shows sample sensory data produced by an edge device embedding two sensors S1, and S2 producing humidity and temperature observations respectively.

TABLE 4.1: Device Data Items

Attribute	Value	Temporal Stamp		Location Stamp		Source
		Format	Value	Format	Value	
Humidity	92 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	Cartesian	(1,2,3)	S1
Temperature	16° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	(2,5,3)	S2
Humidity	94 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	Cartesian	(4,2,7)	S1
Temperature	19.5° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	Cartesian	(5,6,3)	S2
Temperature	21° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	Cartesian	(8,6,3)	S2
Humidity	103 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	(5,2,7)	S1
Temperature	21° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	(8,6,3)	S2
Humidity	104 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	Cartesian	(7,2,7)	S1

4.3.2 Attribute Separation

Since the device can embed various sensors, its internal memory might store different measurements (i.e., attributes such as humidity and temperature in Table 4.1). Therefore, in order to detect redundancies in the data stored locally on the edge device, we start by filtering the data into collections having the same attribute. To illustrate the measurement filtering process, the data shown in Table 4.1 produces two distinct data collections: the first for humidity data (cf. Table 4.2), and the second for temperature data (cf. Table 4.3). Consequently, the data collections are processed separately for data deduplication. Note that the domain expert decides about the selection of attributes to be processed for deduplication.

TABLE 4.2: Humidity Data Collection

Attribute	Value	Temporal Stamp		Location Stamp		Source
		Format	Value	Format	Value	
Humidity	92 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	Cartesian	(1,2,3)	S1
Humidity	94 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	Cartesian	(4,2,7)	S1
Humidity	103 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	(5,2,7)	S1
Humidity	104 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	Cartesian	(7,2,7)	S1

TABLE 4.3: Temperature Data Collection

Attribute	Value	Temporal Stamp		Location Stamp		Source
		Format	Value	Format	Value	
Temperature	16° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	(2,5,3)	S2
Temperature	19.5° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	Cartesian	(5,6,3)	S2
Temperature	21° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	Cartesian	(8,6,3)	S2
Temperature	21° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	(8,6,3)	S2

4.3.3 Pattern Code Generation

Value Pattern Codes

The pattern code generation module transforms ranges of data item values for a given measurement (e.g., humidity, temperature) into interval values that are defined based on reference lookup tables. Edge and sink devices handling the same measurements refer to the corresponding measurement lookup tables (e.g., humidity lookup table, or temperature lookup table), where lookup tables are created based on expert preferences or application requirements. Here, we distinguish between two kinds of lookup tables allowing: (i) disjoint data ranges; and (ii) intersecting data ranges.

TABLE 4.4: Sample Disjoint Value Lookup Tables

(A) Disjoint Humidity Data Ranges

Interval Values	[90 - 96] $\mu g/m^3$	]96 - 104] $\mu g/m^3$	]104 - 110] $\mu g/m^3$
Pattern Code	H1	H2	H3

(B) Disjoint Temperature Data Ranges

Interval Values	[15 - 19] °C	]19 - 24] °C	]24 - 28] °C
Pattern Code	T1	T2	T3

TABLE 4.5: Sample Intersecting Value Lookup Tables

(A) Intersecting Humidity Data Ranges

Interval Values	[90 - 98] $\mu g/m^3$	[94 - 106] $\mu g/m^3$	[102 - 110] $\mu g/m^3$
Pattern Code	H1	H2	H3

(B) Intersecting Temperature Data Ranges

Interval Values	[15 - 20] °C	[18 - 25] °C	[23 - 28] °C
Pattern Code	T1	T2	T3

Disjoint data ranges (cf. Table 4.4) allow simple pattern code generation, yet they produce disconnected pattern codes where values on the range boundaries might be misrepresented (e.g., it is not clear which pattern code can be assigned with values 96.2 $\mu g/m^3$ or 104.7 $\mu g/m^3$ following Table 4.4). Intersecting data ranges (cf. Table 4.5) allow the generation of combined pattern codes when the target value belongs to more than one range (e.g., humidity values 103, 104, and 105 $\mu g/m^3$ belong to both H2 and H3 patter codes following Table 4.5). In this study, we consider intersecting ranges to allow more efficient processing (duplicate candidate filtering module) and more accurate data deduplication (fuzzy redundancy detection module).

Zone Pattern Codes

In case data measurements are produced by mobile sensors (e.g., mobile phones, car sensors), deciding if a pair of data is duplicate or not will involve an extra step: making sure that the two data items are sensed in the same location zone (i.e., in close proximity). Here, we consider that mobile measurements taken at separate location zones represent separate data items, and will not be considered for deduplication. To this end, we introduce zone lookup tables which organize location zones within a connected environment. Our pattern code generation module transforms ranges of location stamps for a given measurement into interval location stamps that are defined based on the zone lookup tables. Edge devices handling the same measurements refer to the corresponding zone lookup tables, where zones can be defined at the level of the network as a whole, or at the level of individual sink or edge nodes, based on the expert and application needs. Table 4.6 shows a sample zone lookup

table describing a sample zoning in our smart hospital example shown in Figure 4.2, where each zone is associated a non-intersecting set of location stamps. Note that a zone can take any shape and size based on the expert and application needs.

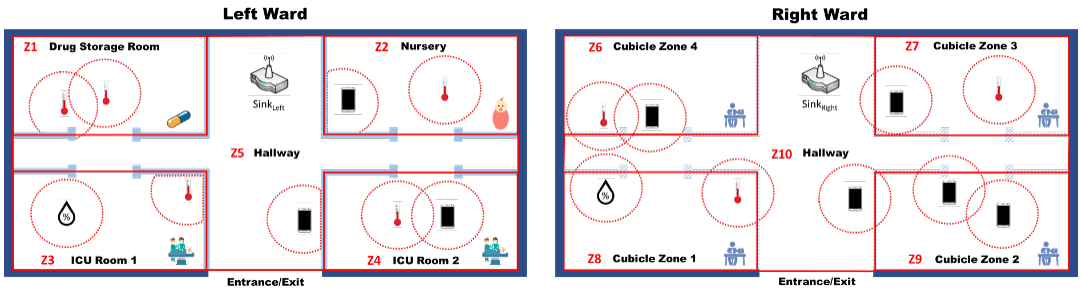


FIGURE 4.2: Smart Hospital Zone Divisions

TABLE 4.6: Zone Lookup Tables

(A) Left Ward Zone Lookup Tables

Label	Location Stamp Interval	Zone Code
Drug Storage Room	$\{p_1, ..., p_k\}$	Z1
Nursery	$\{p_{k+1}, ..., p_m\}$	Z2
ICU Room 1	$\{p_{m+1}, ..., p_n\}$	Z3
ICU Room 2	$\{p_{n+1}, ..., p_o\}$	Z4
Hallway	$\{p_{o+1}, ..., p_p\}$	Z5

(B) Right Ward Zone Lookup Tables

Label	Location Stamp Interval	Zone Code
Cubicle Zone 4	$\{p_{p+1}, ..., p_q\}$	Z6
Cubicle Zone 3	$\{p_{q+1}, ..., p_r\}$	Z7
Cubicle Zone 1	$\{p_{r+1}, ..., p_s\}$	Z8
Cubicle Zone 2	$\{p_{s+1}, ..., p_t\}$	Z9
Hallway	$\{p_{t+1}, ..., p_u\}$	Z10

Combined Pattern Codes

For every data item to be deduplicated, our pattern code generation module produces one (or more) value pattern code(s) and a zone pattern code based on the reference value and zone lookup tables, and then combines them into value-zone pattern code(s) as shown in Tables 4.7 and 4.8 (cf. Algorithm 3). The value-zone codes are used for fast duplicate candidate filtering as described in the following section.

TABLE 4.7: Pattern Codes - Humidity Data Collection

Attribute	Value	Value Pattern Code	Temporal Stamp Format, Value	Location Stamp Format, Value	Zone Pattern Code	Combined Pattern Code	Source
Humidity	92 $\mu g/m^3$	$\{H1\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:00:00	Cartesian (1,2,3)	$\{Z1\}$	$\{H1 - Z1\}$	S1
Humidity	94 $\mu g/m^3$	$\{H1\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:02:00	Cartesian (4,2,7)	$\{Z5\}$	$\{H1 - Z5\}$	S1
Humidity	103 $\mu g/m^3$	$\{H2, H3\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:05:00	Cartesian (5,2,7)	$\{Z5\}$	$\{H2 - Z5, H3 - Z5\}$	S1
Humidity	104 $\mu g/m^3$	$\{H2, H3\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:06:00	Cartesian (7,2,7)	$\{Z5\}$	$\{H2 - Z5, H3 - Z5\}$	S1

TABLE 4.8: Pattern Codes - Temperature Data Collection

Attribute	Value	Value Pattern Code	Temporal Stamp Format, Value	Location Stamp Format, Value	Zone Pattern Code	Combined Pattern Code	Source
Temperature	16° C	{T1}	dd/MM/yyyy hh:mm:ss 10/02/2019 10:01:00	Cartesian (2,5,3)	{Z2}	{T1 − Z2}	S2
Temperature	19.5° C	{T1, T2}	dd/MM/yyyy hh:mm:ss 10/02/2019 10:02:00	Cartesian (5,6,3)	{Z4}	{T1 − Z4, T2 − Z4}	S2
Temperature	21° C	{T2}	dd/MM/yyyy hh:mm:ss 10/02/2019 10:03:00	Cartesian (8,6,3)	{Z4}	{T2 − Z4}	S2
Temperature	21° C	{T2}	dd/MM/yyyy hh:mm:ss 10/02/2019 10:05:00	Cartesian (8,6,3)	{Z4}	{T2 − Z4}	S2

4.3.4 Duplicate Candidate Filtering

Since sensor data items are produced and ordered per sensing time stamp, each data item to be deduplicated is evaluated with its previous one to check if the data is duplicate or not. Our duplicate candidate filtering algorithm is depicted in Algorithm 3. It accepts as input two consecutive data items and produces as output a decision of whether the data items are duplicates, non-duplicates, or candidates for deduplication, based on the following rules: (i) if two data items share one or more value-zone pattern codes, then they are considered duplicates (cf. Algorithm 3, lines 4-5); (ii) if the data items share one or more value-zone pattern codes, they are considered as candidates for deduplication (cf. Algorithm 3 , lines 6-7); and (iii) if the data items do not share any value-zone pattern code, they are considered as non-duplicates (cf. Algorithm 3, lines 8-9).

Algorithm 3: Duplicate Candidate Filtering

Input : $DataItem_1$, $DataItem_2$
Output : $DeduplicationStatus$

```
// Begin algorithm
1  pattern_1 ← pattern code for DataItem_1
2  pattern_2 ← pattern code for DataItem_2
3  interLen ← length of intersection between DataItem_1 and DataItem_2
4  if (pattern_1 = pattern_2 AND interLen = 1) then
5  |  DeduplicationStatus ← Duplicates
6  else
7  |  if (interLen > 1) then
8  |  |  DeduplicationStatus ← Candidates
9  |  else
10 |  |  DeduplicationStatus ← NotDuplicates
11 |  end
12 end
```

Tables 4.9 and 4.10 show the output of the filtering algorithm applied on the input data from Tables 4.7 and 4.8 respectively, where 6 data items are identified as either duplicates/non-duplicates (duplicates are highlited in green and non duplicates are highlighted in red), such that 2 of the original 8 items need to be further considered for fuzzy deduplication (candidates are highlighted in yellow). Depending on the data patterns generated in the target connected environment, duplicate filtering can significantly reduce the number of data items to be processed for fuzzy redundancy detection, thus significantly improving overall processing performance especially at the device level (cf. experimental results in Section 4.4).

TABLE 4.9: Filtering Output - Humidity Data Collection

Attribute	Value	Value Pattern Code	Temporal Stamp Format, Value	Location Stamp Format, Value	Zone Pattern Code	Combined Pattern Code	Source
Humidity	92 $\mu g/m^3$	$\{H1\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:00:00	Cartesian (1,2,3)	$\{Z1\}$	$\{H1-Z1\}$	S1
Humidity	94 $\mu g/m^3$	$\{H1\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:02:00	Cartesian (4,2,7)	$\{Z5\}$	$\{H1-Z5\}$	S1
Humidity	103 $\mu g/m^3$	$\{H2, H3\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:05:00	Cartesian (5,2,7)	$\{Z5\}$	$\{H2-Z5, H3-Z5\}$	S1
Humidity	104 $\mu g/m^3$	$\{H2, H3\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:06:00	Cartesian (7,2,7)	$\{Z5\}$	$\{H2-Z5, H3-Z5\}$	S1

TABLE 4.10: Filtering Output - Temperature Data Collection

Attribute	Value	Value Pattern Code	Temporal Stamp Format, Value	Location Stamp Format, Value	Zone Pattern Code	Combined Pattern Code	Source
Temperature	16° C	$\{T1\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:01:00	Cartesian (2,5,3)	$\{Z2\}$	$\{T1-Z2\}$	S2
Temperature	19.5° C	$\{T1, T2\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:02:00	Cartesian (5,6,3)	$\{Z4\}$	$\{T1-Z4, T2-Z4\}$	S2
Temperature	21° C	$\{T2\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:03:00	Cartesian (8,6,3)	$\{Z4\}$	$\{T2-Z4\}$	S2
Temperature	21° C	$\{T2\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:05:00	Cartesian (8,6,3)	$\{Z4\}$	$\{T2-Z4\}$	S2

4.3.5 Fuzzy Redundancy Detection

Fuzzy Inference Agent

The fuzzy redundancy detection module's overall process is shown in Figure 4.3. It is designed as a fuzzy agent which accepts as input data items that are candidates for redundancy detection, and then produces as output their deduplication status (i.e., duplicates or non-duplicates).

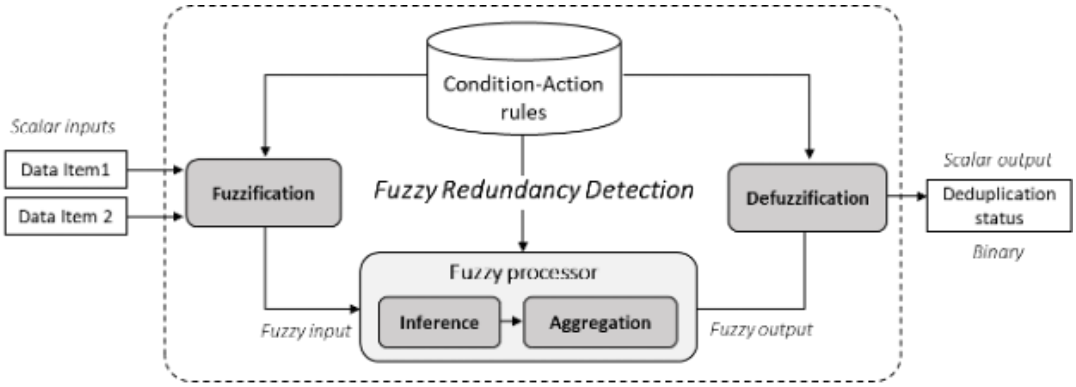
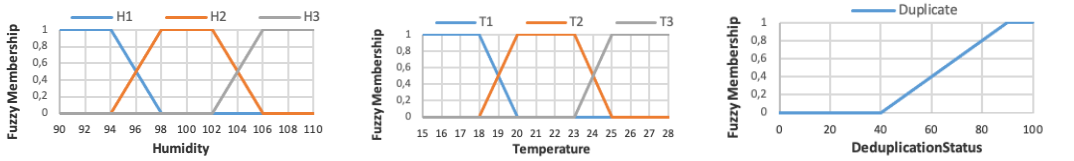


FIGURE 4.3: Fuzzy Redundancy Detection Overview

Fuzzification: First, the scalar data item values are fuzzified, producing linguistic values associated with fuzzy membership degrees (e.g., humidity value 103 $\mu g/m^3$ becomes 75% H2 and 25% H3 following Figure 4.4). The fuzzy partitions for every measurement are defined based on the corresponding lookup table ranges, where the fuzzy membership functions can be defined following the expert and application needs. Figure 4.4a and 4.4b show the fuzzy partitions for humidity and temperature measurements which we adopt in our motivating scenario . The same partitions are utilized to fuzzify both input data values associated with the same measurement. The output deduplication status variable represents a percentage value, and

is depicted in Figure 4.4c using one membership function varying from 0-to-100% duplication.



(A) Input Humidity Fuzzy Partitions cf. Table 4.4a (B) Input Temperature Fuzzy Partitions cf. Table 4.4b (C) Output Deduplication Status Fuzzy Partitions

FIGURE 4.4: Fuzzy Partitions Using The Trapezoidal Function

Condition-action rules: As for the fuzzy agent's condition-action rules, they reflect the common sense logic applied by an domain expert to determine whether two data items are duplicates or not, based on their measurement's look-up tables. We provide below the set of condition-actions rules that we define for the humidity and temperature measurements following our application scenario:

Rule 1

```
IF (Humidity_DataItem1 is H1) AND (Humidity_DataItem2 is
    ↪ H1) THEN DedupStatus is Duplicate
```

Rule 2

```
IF (Humidity_DataItem1 is H2) AND (Humidity_DataItem2 is
    ↪ H2) THEN DedupStatus is Duplicate
```

Rule 3

```
IF (Humidity_DataItem1 is H3) AND (Humidity_DataItem2 is
    ↪ H3) THEN DedupStatus is Duplicate
```

Rule 4

```
IF (Temp_DataItem1 is T1) AND (Temp_DataItem2 is T1) THEN
    ↪ DedupStatus is Duplicate
```

Rule 5

```
IF (Temp_DataItem1 is T2) AND (Temp_DataItem2 is T2) THEN
    ↪ DedupStatus is Duplicate
```

Rule 6

```
IF (Temp_DataItem1 is T3) AND (Temp_DataItem2 is T3) THEN
    ↪ DedupStatus is Duplicate
```

Inference: Fuzzy inference consists in applying the concerned condition-action rules on the fuzzified data in order to produce fuzzy outputs. The logical connectors in the condition-action rules are translated into mathematical formulas that operate on the fuzzy data. For instance, the AND fuzzy logic operator can be any t-norm function, including min (minimum) which is commonly adopted in the literature. Also, THEN can be a number of inference functions including Zadeh's implication (traditional Boolean implication formula), Mamdan's implication (a simplified version of Boolean implication, cf. Formula 4), or Larsen's implication (which comes down to a form of arithmetic multiplication) [61]. In our agent, we adopt Mamdani's implication operator as the default inference function given its common usage in the literature [62, 63]. Yet the aforementioned inference formulas are available through the implemented system, and can be activated following the expert's preferences.

Aggregation: It allows grouping the outputs of multiple inference operations executed on multiple condition-action rules, in order to produce on single fuzzy output result. Multiple mathematical operations can be utilized to simulate fuzzy aggregation, including maximization (aggregating by electing the fuzzy result with the highest membership degree), bounded sum (aggregating by summing the fuzzy membership degrees of the different results, as long as the sum does not surpass 100% membership), and weighted sum (assigning different weights to different inference results, highlighting the importance of different condition-action rules on the decision making process). In our agent, we adopt the maximization aggregation function (Formula 5) given its common usage in the literature [61, 63]. Yet any aforementioned formula can be utilized following the expert's preferences.

Deduplication : It allows transforming the fuzzy output produced by the aggregation function into a crisp output that represents the final result of the agent. Multiple mathematical operations can be used to perform defuzzification, including center of gravity (using the barycenter formula to pinpoint the crisp center of the fuzzy aggregated result), maximum to the left (choosing the smallest crisp value from the aggregated result that has the highest membership degree), and maximum to the right (choosing the highest crisp value that has the highest membership degree). In our agent, we adopt center of gravity (Formula 6) given its common usage in the literature [61, 63]. Yet any aforementioned formula can be utilized following the expert's preferences.

Note that our framework is flexible in allowing experts to apply other fuzzy inference, aggregation, or defuzzification functions of their choosing.

Mamdani's implication: *Given fuzzy sets* f_1, f_2 *:*

$$f_1 \Rightarrow_{Mamdani} f_2 \equiv f_1 \wedge f_2 \equiv min(f_1, f_2) \tag{4.1}$$

where $\wedge$ *is the AND fuzzy logic operator*[1]

Maximization aggregation: *Given fuzzy sets* $f_1, f_2, \ldots, f_n$*:*

$$F_{Agg} = F_{Max} = max(f_1, f_2, \ldots, f_n) \tag{4.2}$$

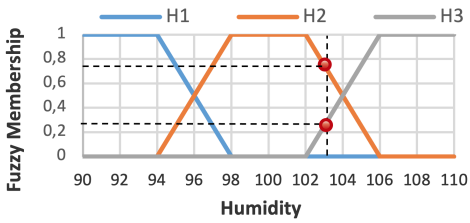
Center of gravity defuzzification: *Given aggregate fuzzy set* F_{Agg}*:*

$$x = \frac{\int x \times F_{Agg}(x) \times dx}{\int F_{Agg}(x) \times dx} \tag{4.3}$$

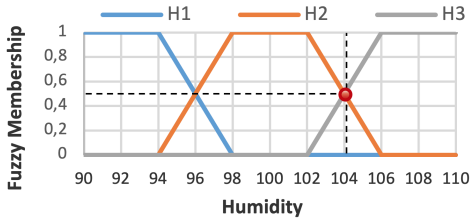
Computation Example

We consider in Tables 4.9 and 4.10 two cases for humidity and temperature measurements studied in our motivation scenario. The detailed computation process for humidity is described in the following use case. A similar computation process for temperature is provided in Appendix A. For the humidity case, the agent recommends that input 103 $\mu g/m^3$ and 104 $\mu g/m^3$ data values are duplicates with a 76% fuzzy membership degree, which seems reasonable given the humidity lookup tables and value ranges defined previously in Table 4.5a (H2 and H3 fuzzy partitions intersect between [102, 106] $\mu g/m^3$, where 103 is much closer to the 102 $\mu g/m^3$ boundary of H2 than to the 106 $\mu g/m^3$ boundary of H3, but also 103 $\mu g/m^3$ and 104 $\mu g/m^3$ are close to each other). The fuzzy inference agent produces recommendations that simulate the domain expert's deduplication capability, and behaves following the expert's design choices and needs (cf. experiments in Section 4.4).

Fuzzification: Given case 1's input data: $Humidity\ DataItem1 = 103\mu g/m^3$ and $Humidity\ DataItem2 = 104\mu g/m^3$, we compute the corresponding fuzzy membership values (cf. Figure 4.5) following the humidity fuzzy functions in Figure 4.4a (reported below):



(A) For $Humidity\ DataItem1$: $fH1(103) = 0, fH2(103) = 0.75, and fH3(103) = 0.25$



(B) For $Humidity\ DataItem2$:$fH1(104) = 0, fH2(104) = 0.5, and fH3(104) = 0.5$

FIGURE 4.5: Fuzzification of Humidity $DataItem1$ and $DataItem2$

[1]The AND fuzzy logic operator can be any t-norm function, including min which is commonly adopted in the literature.

Condition-Action Rules: Based on the input membership values, the following condition-action rules are invoked:

Rule 2

```
IF (Humidity_DataItem1 is H2) AND (Humidity_DataItem2 is
    ↪ H2) THEN DedupStatus is Duplicate
```

Rule 3

```
IF (Humidity_DataItem1 is H3) AND (Humidity_DataItem2 is
    ↪ H3) THEN DedupStatus is Duplicate
```

Inference: Applying Mamdani's inference mechanism:

- **Rule 2:** Executing the AND fuzzy operator:
  - $min(f(103)_{H2}^{Humidity\_DataItem1}, f(104)_{H2}^{Humidity\_DataItem2}) = min(0.75, 0.5) = 0.5$
- Executing the implication fuzzy operator:
  - $f_{Rule2} = min(0.5, Duplicate(DedupStatus))$

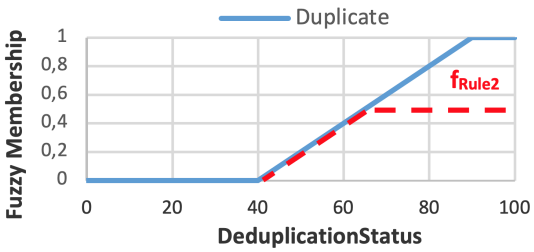


FIGURE 4.6: Inference - Rule2

- **Rule 3:** Executing the AND fuzzy operator:
  - $min(f(103)_{H3}^{Humidity\_DataItem1}, f(104)_{H3}^{Humidity\_DataItem2}) = min(0.25, 0.5) = 0.25$
- Executing the implication fuzzy operator:
  - $f_{Rule3} = min(0.25, Duplicate(DedupStatus))$

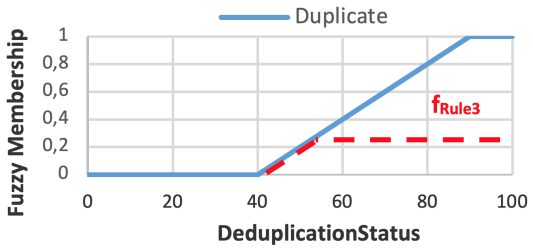


FIGURE 4.7: Inference - Rule3

Aggregation: By applying the maximization aggregation function, $F_{agg} = F_{max} = max(f_{Rule2}, f_{Rule3})$, the agent produces the fuzzy coverage areas subsumed by the inference membership functions (represented in transparent grey color in the below graph).

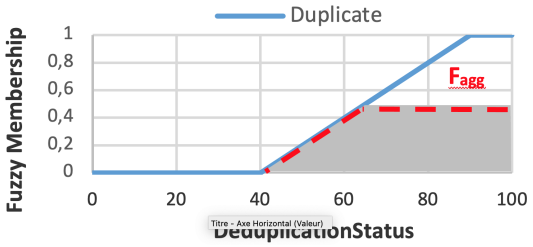


FIGURE 4.8: Aggregation

Defuzzification: The center of gravity defuzzification function is applied on the fuzzy coverage area to compute the corresponding center of gravity point (represented as a red dot in the aggregation graph), and then identify the corresponding deduplication status (on the x axis) the agent's output = 76%.

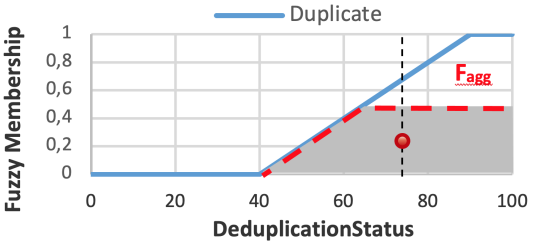


FIGURE 4.9: Defuzzification

Result: Depending on the domain expert or system deduplication threshold, a decision is made whether the two data items are duplicates or not. Given $dedup_{threshold} = 75\%$ in our running example, and since the output of the defuzzification step is $76\% \geq dedup_{threshold}$, the agent's final output becomes: $dedupStatus = duplicates$.

Given our running example data from Table 4.9, the identified humidity redundancies following the fuzzy redundancy detection process are shown in Table 4.11 where duplicates share the same row color.

TABLE 4.11: Result - Humidity Data Collection

Attribute	Value	Value Pattern Code	Temporal Stamp Format, Value	Location Stamp Format, Value	Zone Pattern Code	Combined Pattern Code	Source
Humidity	92 $\mu g/m^3$	$\{H1\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:00:00	Cartesian (1,2,3)	$\{Z1\}$	$\{H1-Z1\}$	S1
Humidity	94 $\mu g/m^3$	$\{H1\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:02:00	Cartesian (4,2,7)	$\{Z5\}$	$\{H1-Z5\}$	S1
Humidity	103 $\mu g/m^3$	$\{H2, H3\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:05:00	Cartesian (5,2,7)	$\{Z5\}$	$\{H2-Z5, H3-Z5\}$	S1
Humidity	104 $\mu g/m^3$	$\{H2, H3\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:06:00	Cartesian (7,2,7)	$\{Z5\}$	$\{H2-Z5, H3-Z5\}$	S1

4.4 FREDD Evaluation

4.4.1 Complexity Evaluation

The overall time complexity of our FREDD approach simplifies to: $\mathcal{O}(N)$ where N designates the number of data items considered in an edge device. Complexity is evaluated as the sum of the complexities of the main modules of FREDD: the complexities of FREDD's modules are linear w.r.t. the number of data items being processed at the edge, and simplify in the worst case scenario to $\mathcal{O}(N)+\cdots+\mathcal{O}(N)$, which comes down to an overall $\mathcal{O}(N)$.

4.4.2 Experimental Evaluation

We have implemented our FREDD framework as a web-based application, using methods from the jFuzzyLogic open source library [64, 65] in implementing our fuzzy logic agent, to allow easy manipulation for domain experts in operating and evaluating the system[2]. We have empirically tested the different components of our system using multiple sets of experiments which we categorize in two main groups: (i) quality evaluation: comparing deduplication accuracy, data reduction ratio, size of transmitted data, and size of stored data in order to evaluate deduplication quality; and (ii) performance evaluation: comparing the time performance of the different components of the system, in order to evaluate its time complexity. We first start by describing our test data and experimental metrics, before we present our empirical results. The system implementation, experimental datasets, and test results are available online[3].

Experimental Test Data

We build two datasets for edge device, mobile device, and sink device measurements collected from the Intel Lab Berkeley dataset [66] obtained from 54 Micra2Dot sensors depicted in the Figure 4.10. Sensors provide weather data including temperature, humidity, light, (and voltage at the time of the sensor reading), as well as the list of Cartesian coordinates for each of the 54 sensors, and the time when each data measurement is collected. We consider humidity and temperature measurements in our experimental evaluation and describe our datasets below:

[2]On the server-side, we adopt a three-layer architecture consisting of: i) a Web API layer that allows client-side applications to communicate with the server to request data, services and to define all the domain expert parameters such as the deduplication threshold, the value and zone lookup tables, the different fuzzy parameters, etc.; ii) a Business Logic layer where FREDD's main decision making processes are implemented based on the different parameters the expert provided; and iii) a Data Access layer where data storage and retrieval take place. Every layer is internally designed in a modular way to allow for separate testing and evaluation of every module.

[3]http://sigappfr.acm.org/Projects/FREDD/

- Dataset 1: Static edged device dataset – It consists of 20k humidity and temperature data measurements collected from sensor S1 on 28/2/2004.
- Dataset 2: Mobile edge device dataset - We assume a mobile device M1 and construct its dataset from the static Micra2Dot sensors as follows: (i) humidity and temperature data for M1 is collected from nine sensors: S1-to-S4 and S6-to-S10 between 28/2/2004 and 29/2/2004; (ii) data collected from all these sensors is first ordered chronologically by date and time, and then filtered to simulate the following path of the mobile sensor (S1 $\rightarrow$ S2 $\rightarrow \ldots \rightarrow$ S9 $\rightarrow$ S10 $\rightarrow$ S9 $\rightarrow$ S8 $\ldots \rightarrow$ S1); and (iii) mobile sensor M1 collects from each location a random number of data measurements[4], resulting in a dataset of 3,416 entries. The resulting dataset simulates the behavior of a mobile sensor where the location of each data measurement is that of the source static sensor collecting it in the Micra2Dot schematic.

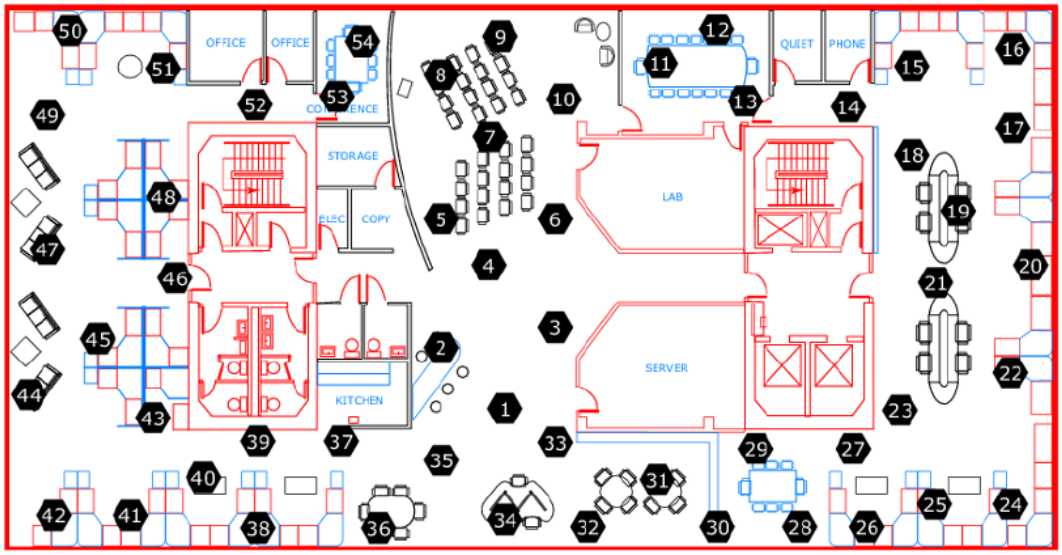


FIGURE 4.10: Schematic of the Intel Lab Berkeley Micra2Dot sensors

Evaluation Metrics

We utilize four evaluation metrics to evaluate FREDD's deduplication effectiveness. At the edge device, we utilize (i) deduplication accuracy; and (ii) data reduction percentage. We describe the two metrics below.

Deduplication accuracy is defined as a time series similarity between the original data and the deduplicated data, after modifications have been applied on the deduplicated data set in order to reconstruct a set that has the same dimension (length) of the original one [31]. More formally, given $TS_o = [(t_1, v_{o_1}), (t_2, v_{o_2}), \ldots, (t_n, v_{o_n})]$ as the time series representation of the original data where n is the length of the data, and $TS_d = [(t_1, v_{d_1}), (t_2, v_{d_2}), \ldots, (t_m, v_{d_m})]$ as the time series representation of the deduplicated data where m is the length of the deduplicated data such that $m < n$ and $TS_d \in TS_o$, we generate $TS_r = [(t_1, v_{r_1)}, (t_2, v_{r_2}), \ldots, (t_n, v_{r_n})]$ as the reconstructed time series from the deduplicated data where the missing (deduplicated) values are padded to reach the same dimensionality of the initial data.

[4]We use a random integer between 1 and 5, where a small integer will increase the chance of changing zone pattern codes between two consecutive data measurements, emphasizing the idea of mobility.

For instance, given $TS_o = [(t_0, 16°C), (t_1, 19.5°C), (t_2, 21°C), (t_3, 21°C)]$ and $TS_d = [(t_0, 16°C), (t_3, 21°C)]$, then $TS_r = [(t_0, 16°C), (t_1, 21°C), (t_2, 21°C), (t_3, 21°C)]$ where the missing values at t_1 and t_2 have been padded by the deduplicated value at t_0. Consequently, deduplication accuracy is measured as the Jaccard similarity coefficient between the original time series and the reconstructed (same dimensionality) time series as follows:

$$acc = \frac{\sum_{i=1}^{n} min(v_{o_i}, v_{r_i})}{\sum_{i=1}^{n} max(v_{o_i}, v_{r_i})} \in [0, 1] \tag{4.4}$$

A good deduplication solution would produce a higher similarity between the original data and the reconstructed data, resulting in higher deduplication accuracy.

Data reduction ratio represents the amount of data that has been eliminated as a result of applying the deduplication process. More formally, it is defined as the ratio of the difference between the original data and the duplicated data:

$$redu = \left(1 - \frac{|TS_o| - |TS_d|}{|TS_o|}\right) \in [0, 1] \tag{4.5}$$

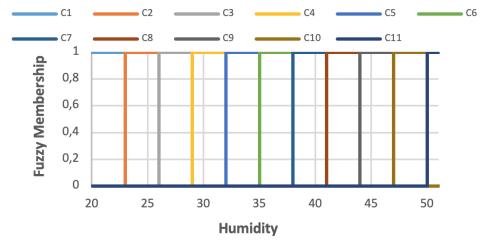
where $|TS_o|$ represents the size of the original data, $|TS_d|$ the size of the deduplicated data, and $\frac{|TS_o|-|TS_d|}{|TS_o|}$ the data saving ratio. A good deduplication solution would produce a lower data saving ratio (i.e., smaller difference between original and deduplicated data size), resulting in a higher data reduction ratio.

Quality Evaluation

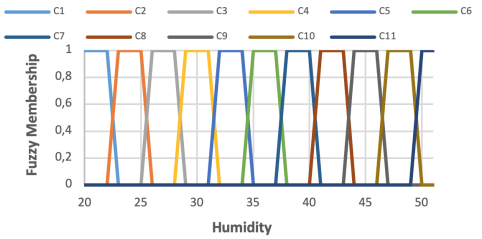
We conduct multiple sets of experiments to evaluate FREDD's deduplication effectiveness considering various parameters and use case scenarios, evaluating: (i) data range overlap size; and (ii) fuzzy deduplication threshold.

Data Range Overlap Size Evaluation: In this experiment, we evaluate the behavior of FREDD's fuzzy redundancy detection process when varying the size of the data overlap between boundary value ranges. This allows the domain expert to easily update the fuzzy membership functions according to the size of the boundary overlapping, and thus allows more flexibility in fine-tuning the solutions' behavior following the expert's needs (cf. Challenge 3 - Chapter 1). We consider four different pattern code fuzzy membership functions as shown in Figure 4.11: (i) rectangular functions with no overlapping boundaries amounting to 0% fuzzy computation; (ii) trapezoidal functions with small overlapping boundaries amounting to 30% fuzzy computation; (iii) trapezoidal functions with large overlapping boundaries amounting to 50% fuzzy computation; and (iv) triangular functions with completely overlapping boundaries amounting to 100% fuzzy computation. Deduplication accuracy and data reduction ratio results, applied on static edge data from dataset 1, are shown in Figure 4.12. Results show that: (i) deduplication accuracy (*acc*) increases; and (ii) reduction ratio (*redu*) decreases when the overlap size between boundary ranges increases. On the one hand, an increase in boundary overlapping leads to more fuzzy duplicate candidates being evaluated and detected by the system. This leads to higher *acc* since the deduplicated values resulting from the fuzzy process will be closer to the original data values within the overlapping boundaries (compared with performing a crisp decision making where the deduplicated values are

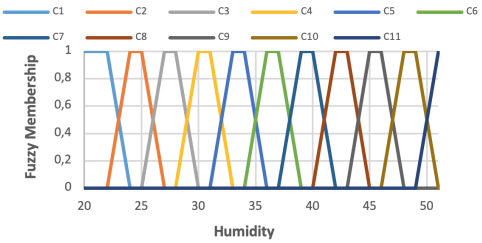
restricted to the crisp boundaries, which are naturally farther away from the original data values within those boundaries). On the other hand, an increase in fuzzy processing leads to a lower *redu* since larger fuzzy ranges allow more candidate data to be considered for redundancy check. This leads to an increase in the amount of data considered for processing and persisting after the deduplication process (in contrast, a larger amount of data that is directly deduplicated following the crisp approach produces a leaner deduplication result, albeit with less accuracy compared with the original data).



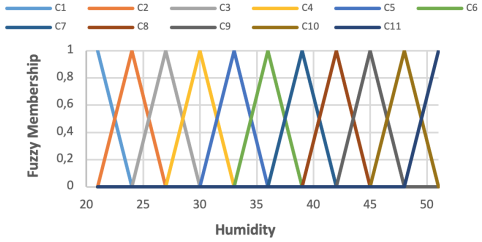
(A) Rectangular with 0% overlap



(B) Trapezoidal with 30% overlap

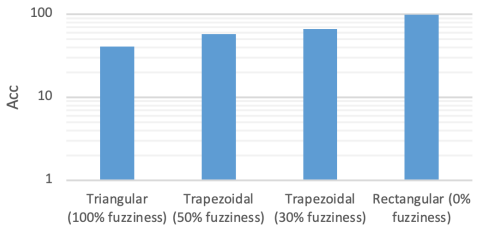


(C) Trapezoidal with 50% overlap

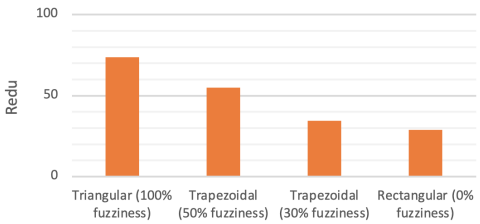


(D) Triangular with 100% overlap

FIGURE 4.11: Different humidity pattern code fuzzy membership functions with different boundary range overlapping sizes



(A) Deduplication accuracy (*acc*)

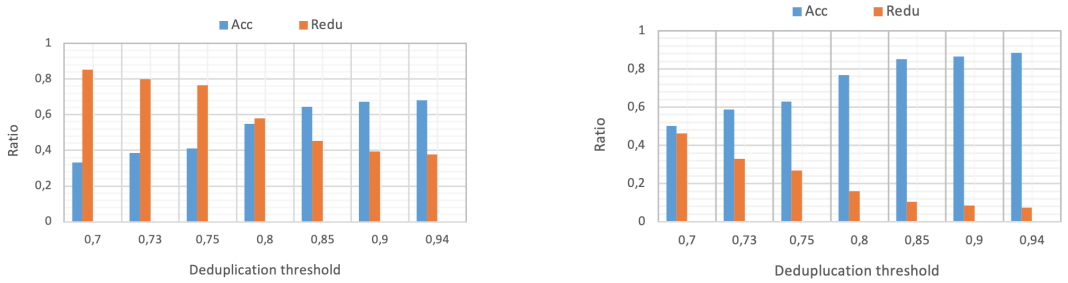


(B) Data reduction ratio (*redu*)

FIGURE 4.12: *acc* & *redu* results when varying the overlapping size of boundary data ranges

Fuzzy Deduplication Threshold Evaluation In this experiment, we vary the fuzzy deduplication threshold, allowing the fuzzy redundancy detection process to decide on the deduplication status of candidate data items, and evaluate FREDD's behavior accordingly. We perform this experiment at the edge node level (cf. Challenge 1 - Chapter 1), and consider the impact of device mobility on the deduplication process. This allows the domain expert to choose a suitable deduplication threshold in order to achieve the desired accuracy and deduplication ratio following the expert's needs (cf. Challenge 3 - Chapter 1). Figure 4.13 shows the results of deduplication quality metrics when applied on static edge data from dataset 1 (Figure 4.13a), and on mobile edge data from dataset 2 (Figure 4.13b), by varying the deduplication

threshold. For both cases, when the threshold increases: (i) *acc* increases while (ii) *redu* decreases. This is due to the fact that a higher deduplication threshold means less candidate pairs are considered for duplication.



(A) Results with static device data from dataset 1

(B) Results with mobile device data from dataset 2

FIGURE 4.13: *acc* & *redu* results with varying fuzzy deduplication thresholds

Performance Evaluation

In addition to testing the quality of our approach in identifying redundant data items and performing deduplication, we also evaluate its efficiency in terms of execution time. Results in Figure 4.14 highlight the linear complexity of FREDD's edge-based deduplication process when varying the number of data items per edge node, reflecting $\mathcal{O}(N)$ time complexity.

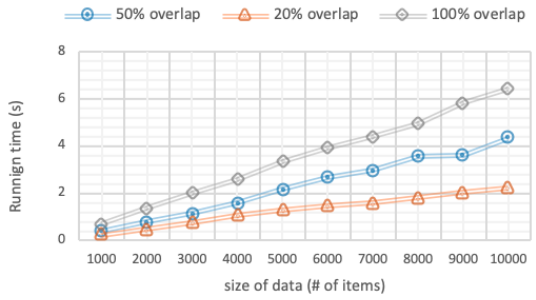


FIGURE 4.14: Edge-level processing time when varying the number of data items

4.5 Recap

In this study, we introduce FREDD: a new approach for Fuzzy Redundancy Elimination for Data Deduplication in a connected environment. FREDD uses simple natural language rules to represent domain knowledge and expert preferences regarding data duplication boundaries. It then applies pattern codes and fuzzy reasoning to detect duplicates on the edge level. Moreover, it is adapted for multiple scenarios, considering both static and mobile devices (cf. Challenge 1 - Chapter 1). Furthermore, it allows users to configure pattern code overlaps and customize the redundancy detection process (cf. Challenge 3 - Chapter 1). We also present experiments on a real-world dataset that highlight FREDD's potential. In the following, we present how FREDD can be used for data redundancy detection at the sink level of a connected environment network since we already covered redundancy detection

at the source in Chapters 3 and 4. In order to do so, we discuss next the constraints that we need to consider in regards to the physical infrastructure, the device capabilities, and the spatial configuration of the environment (cf. Challenge 2 - Chapter 1) to achieve data redundancy at the sink.

Chapter 5

Detecting Data Redundancies at the Sink Level

In this chapter, we focus on detecting data redundancies at the sink level (i.e., core) of a CE. Tackling the problem at the sink level allows the detection of composite or collective redundancies from various edge devices (since edge devices submit their data to the sink). In addition, handling the problem at the sink provides an alternative that alleviates resource consumption on edge devices (where resources are often limited). Moreover, it improves querying the core of the network directly by allowing the retrieval and the processing of redundancy-free sensor observations.

We previously proposed FREDD, a Fuzzy Redundancy Elimination for Data Deduplication framework, that utilizes fuzzy reasoning to accurately and efficiently detect redundancies at the source (i.e., locally on devices). We also demonstrated how FREDD can be used on the edge.

We discuss here the application of FREDD on the sink level of the network, and the requirements that need to be considered to do so (cf. Challenge 2). This entails considering the different separations between individual locations in the environment (e.g., walls, windows, open spaces, curtains) that affect the sensing capabilities of devices (e.g., impact on sensor coverage areas) as well as the distribution/density of sensors and sink nodes (e.g., number of sensors connected to a specific sink).

We discuss four use cases for using FREDD for data redundancy detection at the sink level. We detail the different environmental constraints that play into effect in each use case. Then, we detail the FREDD process at the sink level.

Thereafter, we present an extensive evaluation redundancy detection by presenting the experimental protocol, the conducted experiments, and the obtained results. Finally, we conclude this chapter with a recap and future directions.

Sylvana Yakhni, Joe Tekli, Elio Mansour, Faisal Shahzad, and Richard Chbeir. Fuzzy Redundancy Elimination for Data Deduplication in Connected Environments. Submitted to the Journal of Parallel and Distributed Computing.

5.1 Introduction

In most CE configurations, sensing devices are deployed in the real-world (i.e., the physical environment/infrastructure) and handle data acquisition. Regardless of their capability to exchange data between them, they often end up submitting their data to a central node or base station called the sink. Each connected environment has a specific distribution/density of edge sensing devices that submit their data to particular sink nodes. Also, the number of sink nodes in a network varies depending on the scale of the environment. Furthermore, each environment has a set of known locations and specific spatial constraints that define its constitution (e.g., hard wall separations between locations, soft separations, open spaces) and the spatial relations between locations. All the aforementioned environment features affect the sensing (e.g., sensor coverage areas are affected by walls, producing data from different locations/zones). However, in all cases, data redundancies can occur on the edge of the network (i.e., locally on devices) and on the sink of the network where data is accumulated and new redundancies appear when combining data from multiple sources.

In Chapters 3 and 4, we proposed DRMF and FREDD capable of detecting redundancies at the edge (i.e., locally on devices). However, these approaches might miss some redundancies that do not exist locally but appear globally when combining data from various sources and comparing them together. For instance, two sensing devices located in close proximity might generate a temperature observation each. These observations can be non-redundant locally (i.e., compared to previous values stored locally on the device). However, due to the sensors' proximity, the two temperature observations become redundant on a more global level (e.g., in the zone/area of deployment). This means that redundancy monitoring needs to to take place on all network levels (this includes the edge and core levels). In fact, handling redundancies at different levels provides different advantages and inconveniences. On the one hand, managing redundancies at the edge level (i.e., source) helps alleviate excessive bandwidth usage and unnecessary data storage in central repositories but adds more processing loads on edge devices and does not consider combined redundancies. On the other hand, handling redundancies at the sink level (i.e., core) alleviates resource consumption at the edge and detects combined redundancies, but it taxes the network by allowing the spreading of duplicate unnecessary data throughout the environment.

Therefore, although we proposed two approaches for edge-level data redundancy detection, there is still a need to propose a similar approach for sink-level data redundancy management. Doing so entails considering global features and constraints related to the environment (e.g., different types of physical objects and inter-location separations), spatial configurations (e.g., location maps, inter-location spatial ties), and device capabilities (e.g., sensor coverage areas) as illustrated in Challenge 2 (cf. Chapter 1).

To address the need for sink-level data deduplication, we describe here how our FREDD proposal can be used at the sink level (i.e., at the core of the network). We consider FREDD since it improves on DRMF by considering fuzzy reasoning to improve its accuracy in an efficient manner. We illustrate four use cases that show how we can consider the aforementioned environmental features (regarding the physical objects, spatial constraints, and sensor coverage areas) to detect combined

redundancies at the sink. Then, we detail the sink-level deduplication process and evaluate it.

The remainder of this Chapter is organized as follows. Section 5.2 provides four use cases for redundancy detection at the sink level to showcase the different features that need to be considered. Then, Section 5.3 describes the FREDD process at the sink level and details each of its modules. Section 5.4 provides a complexity evaluation of FREDD when applying it at the sink as well as an extensive experimentation that assesses the accuracy and performance of this particular proposal. Finally, Section 5.5 summarizes the chapter.

5.2 Sink Level Use Cases

Generally, an edge device (made of one or multiple sensors) can cover an observation if it occurs within its coverage area (i.e., sensing range), where every event that takes place in this area can be detected by the device. The coverage area of an edge device is usually defined by the device manufacturer following its sensor(s) specifications. As for sink devices, their coverage areas are usually defined by experts based on the sinks' connectivity in the environment. A sink device's coverage area can be defined as one or multiple non-overlapping zones, following the network designer's needs (cf. Figure 5.1). In addition, sink-level zones can be hard-separated or soft-separated. With hard-separated zones, a sensor's coverage area lies in one single zone from which it can collect data (e.g., camera sensors separated by walls, cf. Figure 5.1a). With soft-separated zones, a sensor's coverage area spans more than one zone allowing the sensor to collect data from multiple zones simultaneously (e.g., camera sensors separated by glass doors, cf. Figure 5.1b).

(A) Hard Zone Separations

(B) Soft Zone Separations

FIGURE 5.1: Examples of sink node coverage areas, with multiple zones including hard and soft separations

As a result, we consider and discuss four different sink-level deduplication use cases summarized in Table 5.1: (i) zone-based with hard separations; (ii) zone-and-coverage based with hard separations; (iii) zone-based with soft separations; and (iv) zone-and-coverage based with soft separations.

TABLE 5.1: Sink-level deduplication uses cases

Use Cases	Considers Sink Zones	Considers Sensor Coverage Areas	Hard Separations Between Zones	Soft Separations Between Zones
Case 1	✓	✗	✓	✗
Case 2	✓	✓	✓	✗
Case 3	✓	✗	✗	✓
Case 4	✓	✓	✗	✓

5.2.1 Case 1: Zone-based with Hard Zone Separations

In this case, we assume: (i) the sink node coverage zones are hard-separated; and (ii) data from multiple sensors are considered for deduplication if the two sensors are located in the same zone (cf. Figure 5.1a where data from sensors S1-and-S2 are considered for deduplication, likewise for data from sensors S3-and-S4). While it seems simple and straightforward, yet this use case neglects the issue of edge device (sensor) coverage area. In other words, this use case might not be entirely practical, since sensors might be located in the same zone but their coverage areas do not overlap (e.g., the case of sensors S3-and-S4). For example, two cameras might be located next to each other in the same room, but each camera covers its own corner in the room. In such situations, sensor will be producing separate data feeds which are not combined and deduplicated at the sink node, since they describe different things. This can be handled in the following use case #2.

5.2.2 Case 2: Zone-and-Coverage based with Hard Zone Separations

In this case, we assume: (i) the sink node coverage zones are hard-separated; and (ii) data from multiple sensors are considered for deduplication if (1) the sensors collect data from the same zone (i.e., their coverage areas are included in the same zone); and (2) the sensors' coverage areas are largely overlapping (e.g., sensors S1-and-S2 in Figure 5.1a. For instance, the data feeds of two cameras located in the same room and covering largely overlapping areas of the room will be considered for deduplication at the sink node. Yet, if the cameras' coverage areas do not largely overlap, their data feeds will be processed separately and will not be considered for deduplication at the sink (cf. sensors S3-and-S4 in Figure 5.1a). Deciding whether two coverage areas largely overlap or not is done by evaluating the spatial topological relations between the areas (e.g., equal, overlap, and, disjoint, cf. Section 5.3).

5.2.3 Case 3: Zone-based with Soft Zone Separations

In this case, we assume: (i) the sink node coverage zones are soft-separated; and (ii) data from multiple sensors are considered for deduplication if the sensors collect data from the same zone, i.e., if their coverage areas are included in or largely overlap with the same zone (e.g., sensors S5-and-S6 and sensors S7-and-S8 in Figure 5.1b). Deciding on coverage area-zone inclusion and overlapping is done by evaluating the spatial topological relations between the areas and the zones (similarly to area-area topological relations, cf. Section 5.3).

5.2.4 Case 4: Zone-and-Coverage based with Soft Zone Separations

In this case, we assume: (i) the sink node coverage zones are soft-separated; and (ii) data from multiple sensors are considered for deduplication if (1) the sensors collect data from the same zone (their coverage areas are entirely included in or largely overlap with the same zone); and (2) the sensors' coverage areas are largely overlapping, e.g., sensors S5-and-S6 in Figure 5.1b). For instance, the data feeds from two temperature sensors collecting data from the same room such that their coverage areas do not overlap (e.g., sensors S7-and-S8 in Figure 5.1b) will not be considered for deduplication at the sink node.

5.3 FREDD Sink Level Process

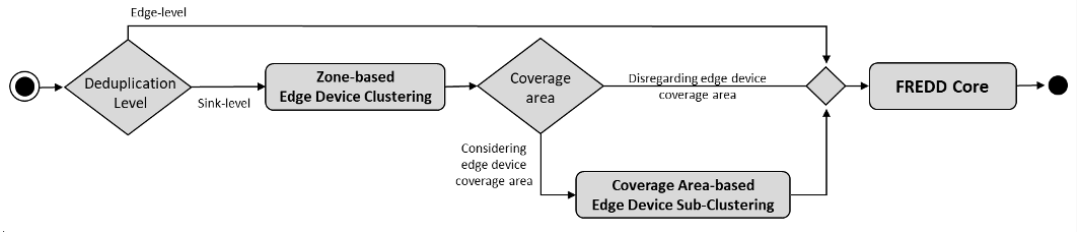
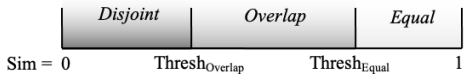


FIGURE 5.2: FREDD's sink-level deduplication process

The overall process of sink-level deduplication using FREDD is depicted in Figure 5.2. We start by performing zone-based edge device clustering to group together edge devices (sensors) bellowing to the same zone. These are the sensors located within the same hard-separated zones (following cases 1-and-2), or the sensors which coverage areas are included in or largely overlap with the soft-separated zones (following cases 3-and-4). Deciding whether an edge device coverage area and a sink device (soft-separated) zone overlap or not is achieved by evaluating the spatial topological relations between them (e.g., equal, overlap, and, disjoint), which comes down to evaluating their geometric similarity in a referential (e.g., Euclidian) geometric space, formally [14, 67]:

$$Sim(area, zone) = \frac{|Intersection(area, zone)|}{min(|area|, |zone|)} \tag{5.1}$$

The amount of overlapping between an area and a zone is controlled by the expert through a dedicated similarity threshold (cf. Figure 5.3a, e.g., $Sim(area, zone) \geq Thresh_{Overlap}$ means the sensor coverage area largely overlaps with the sink zone).



(A) Using a symmetric similarity measure

(B) Using an asymmetric similarity measure

FIGURE 5.3: Basic spatial topological relationships following [14, 67]

Similarly, the inclusion relation between and area and a zone is evaluated using an asymmetric version of the geometric similarity measure, more formally [14, 68] where the inclusion relation occurs if $Sim_{Asym}(area, zone) \geq Thresh_{Include}$ (cf. Figure 5.3b):

$$Sim(area, zone) = \frac{|Intersection(area, zone)|}{|area|} \tag{5.2}$$

Consequently, in cases 2-and-4 where edge device coverage areas are taken into account, we perform area-based edge device sub-clustering, by evaluating edge device (sensor) coverage area similarity (cf. Formula 5.3) and grouping together edge devices having largely overlapping coverage areas. The amount of overlapping between two areas to be considered eligible for sink-level deduplication – is controlled by the expert through a dedicated similarity threshold (cf. Figure 5.3a, e.g., $Sim(area1, area2) \geq Thresh_{Overlap}$ then coverage areas largely overlap and their edge devices are eligible for sink-level deduplication).

$$Sim(area_1, area_2) = \frac{|Intersection(area_1, area_2)|}{min(|area_1|, |area_2|)} \tag{5.3}$$

After all edge devices (sensors) have been clustered and sub-clustered following their target use cases, data from the final clusters are run separately through the FREDD framework to perform the deduplication process.

5.4 FREDD Sink Level Evaluation

5.4.1 Complexity Evaluation

The overall time complexity of our FREDD approach simplifies to: $\mathcal{O}(N \times E^2)$ where N designates the number of data items considered per edge device, and E the number of edge devices considered per sink node. Complexity is evaluated as the sum of the complexities of the main modules of FREDD, considered both edge-level and sink-level processing:

- For sink-level deduplication: the complexities of FREDD's modules are linear w.r.t. the number of data items being processed at the sink, and simplify in the worst case scenario to $\mathcal{O}(N) + \cdots + \mathcal{O}(N)$, which comes down to an overall $\mathcal{O}(N)$.
- For edge-and-sink level deduplication: we consider: (i) the number of edge devices E per sink node, where the system requires worst case $\mathcal{O}(((E \times (E - 1))/2)$, and simplifies to $\mathcal{O}(E^2)$ considering all edge nodes are present in the same sink zone and need to be compared together pair-wise to identify their coverage area intersections; (ii) the number of sink nodes S (i.e., number of zones) in the environment, where the system requires worst case $\mathcal{O}(E \times S)$ to compare every edge node with every sink zone to identify their area-zone intersections. As a result, FREDD's overall complexity when performing edge-and-sink level deduplication comes down to $\mathcal{O}(N \times (E^2 + E \times S))$ which simplifies to $\mathcal{O}(N \times E^2)$ since E is generally much larger than S.

5.4.2 Experimental Evaluation

We evaluate the accuracy and efficiency of FREDD on the sink in a similar manner to its evaluation on the edge (cf. Chapter 4 - Section 4.4). We describe first the dataset used in the experiments. Then, define the metrics that we used for the evaluation. Finally, we detail the experiments and results.

Experimental Test Data

We build three datasets for edge device, mobile device, and sink device measurements collected from the Intel Lab Berkeley dataset that we previously used for the evaluation of FREDD on the edge level. To do so we generated two datasets (dataset 1 and 2) to evaluate static and mobile edge-level deduplication. Here, we introduce a third dataset (dataset 3) to evaluate FREDD on the sink level:

- Dataset 3: Sink device dataset – We assume a sink device Sink1 and construct its dataset from the static Micra2Dot sensors as follows: (i) humidity and temperature data for Sink1 is collected from the following nine sensors (used previously to create Dataset 2): S1-to-S4 and S6-to-S10 between 28/2/2004 and

29/2/2004; (ii) data is ordered chronologically by date and time producing a dataset of 31,135 data entries. The resulting dataset simulates the behavior of a sink device collecting data from 9 different edge devices, where the location of each data measurement is that of the source static sensor collecting it in the Micra2Dot schematic.

Evaluation Metrics

In order to evaluate our proposal at the sink we use the same metrics (i.e., deduplication accuracy, and data reduction ratio) defined in the experimentation of FREDD on the edge (cf. Chapter 4). Moreover, we consider two additional metrics that are relevant for evaluating FREDD on the sink level, or edge-and-sink level:

Size of transmitted data $(|data_{Trans}|)$ represents the size of the data transmitted from the edge devices to the sink device. A good deduplication solution would reduce both the size of data transmitted over the network in order to gain in network bandwidth.

Size of stored data $(|data_{Stored}|)$ represents the size of the data stored at the sink device[1]. A good deduplication solution would reduce the size of the data stored at the sink in order to gain in processing efficiency, speed, and throughput at the sink level.

Quality Evaluation

We conduct multiple sets of experiments to evaluate FREDD's deduplication effectiveness considering various parameters and use case scenarios, evaluating: (i) fuzzy deduplication threshold; (ii) sink zone granularity; (iii) number of edge devices connected to the sink; and (iv) sensor coverage area size. We also conduct a (v) baseline comparison evaluating FREDD's deduplication quality compared with its most recent alternatives. We describe the experiments and their results in the below subsections.

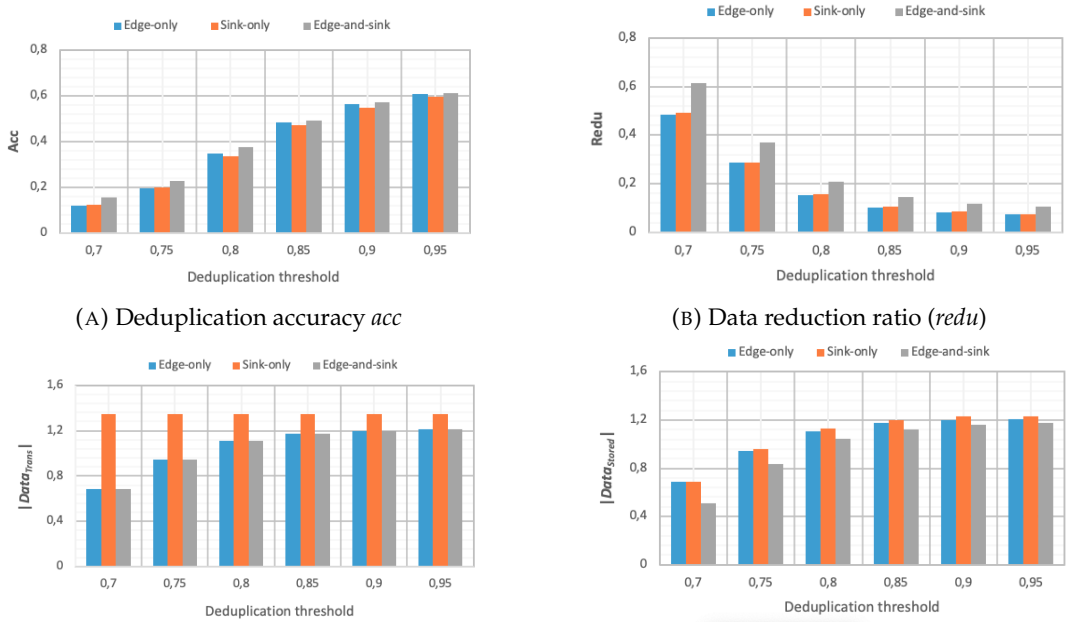
Fuzzy Deduplication Threshold Evaluation In this experiment, we vary the fuzzy deduplication threshold, allowing the fuzzy redundancy detection process to decide on the deduplication status of candidate data items, and evaluate FREDD's behavior accordingly. We perform this experiment at both edge node and sink node levels (cf. Challenges 1 and 2 - Chapter 1). This allows the domain expert to choose a suitable deduplication threshold in order to achieve the desired accuracy and deduplication ratio following the expert's needs (cf. Challenge 3 - Chapter 1). Figure 5.4 shows the results of deduplication quality metrics when applied on data sink device data from dataset 3 by varying the deduplication threshold. When the threshold increases: (i) *acc* increases while (ii) *redu* decreases. This is due to the fact that a higher deduplication threshold means less candidate pairs are considered for duplication. We highlight the following observations:

- While *acc* increases and *redu* decreases when increasing the threshold for the different deduplication methods, we note that the *redu* is relatively higher

[1]We follow the typical sensor network setup where edge devices do not perform any long-term data storage.

when deduplicating at both the edge-and-sink level, compared with edge-only and sink-only deduplications.

- The size of data transmitted to the sink ($|data_{Trans}|$) and the size of data stored at the sink ($|data_{Stored}|$) are both increased with the increase in deduplication threshold. This is mainly due to the decrease in *redu*, resulting in more data being sent and processed at the sink node.
- $|data_{Trans}|$ levels are equal when deduplicating at edge-only and at edge-and-sink, and is less than the $|data_{trans}|$ level when deduplicating at sink-only, since data in the latter case are sent directly from the edge to the sink without edge-level deduplication.
- $|data_{Stored}|$ is smallest in case of deduplication at edge-and-sink, compared with edge-only and sink-only, since deduplication is performed at two levels.



(A) Deduplication accuracy *acc*

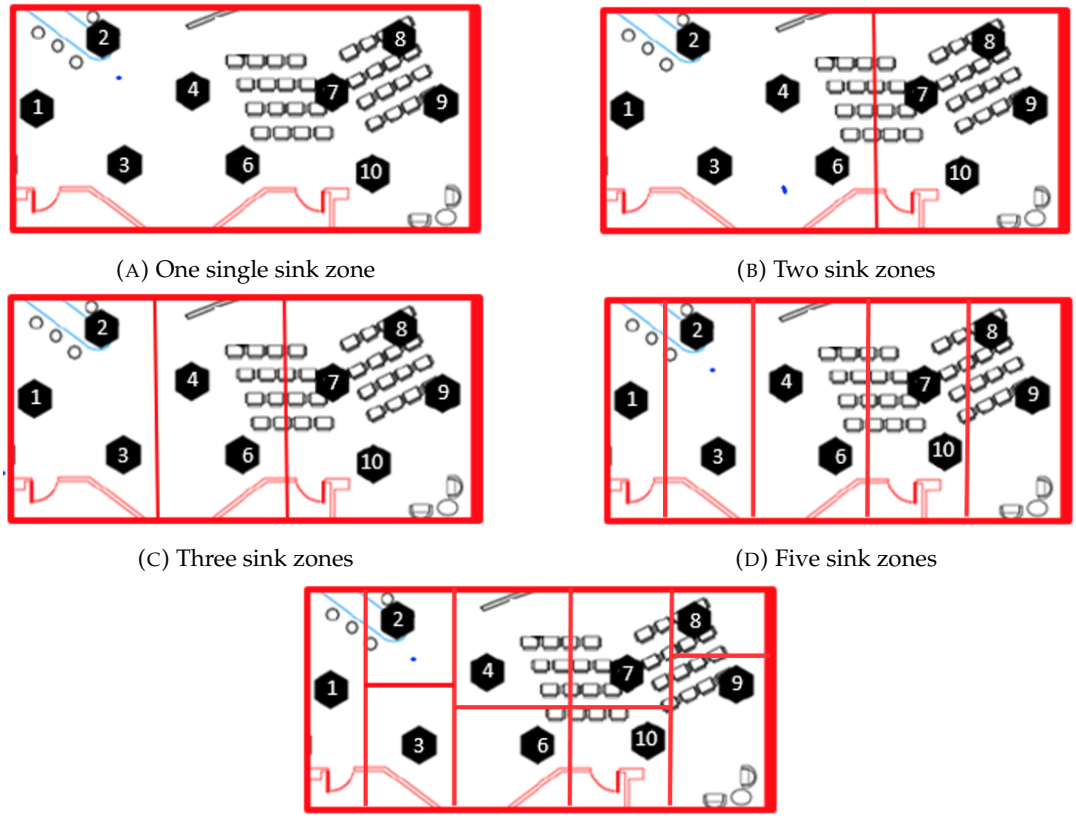
(B) Data reduction ratio (*redu*)

(C) Transmitted data size to sink ($|data_{Trans}|$)

(D) Stored data size at sink ($|data_{Stored}|$)

FIGURE 5.4: Deduplication quality metrics obtained with varying fuzzy deduplication thresholds

Sink Zone Granularity Evaluation In this experiment, we evaluate the impact of sink zone granularity on the deduplication process. Consider the sample zone granularity configurations shown in Figure 5.5. We first consider the area shown in Figure 5.5a as one single zone including 9 sensor devices, and then gradually divide it into smaller non-overlapping zones: 2, 3, 5, and 9 zones respectively (Figures 5.5b to 5.5e). We perform this experiment considering deduplication at the edge and sink levels (cf. Challenges 1 and 2 - Chapter 1). It also highlights the domain expert's ability to divide the connected environment into different sink zone granularities in order to achieve the desired behavior based on the expert and application needs (cf. Challenge 3 - Chapter 1).



(A) One single sink zone

(B) Two sink zones

(C) Three sink zones

(D) Five sink zones

(E) Nine (individual device) sink zones

FIGURE 5.5: Sample sink zone granularities using an extract of our reference Micra2Dot dataset (cf. Figure 4.10)

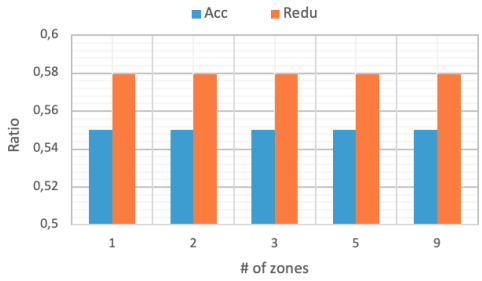
Figures 5.6 and 5.7 show the results of deduplication quality metrics when applied on static edge data from dataset 1 (Figure 5.6), and on mobile edge data from dataset 2 (Figure 5.7), by varying the number of zones in a sink coverage area. We highlight the following observations:

- In the case of static devices (where nodes are not changing zones over time), results show that an increase in zone granularity does not have any impact on the deduplication results (deduplication metrics remain unaffected).
- In the case of mobile device (where nodes are changing zones over time), results show that an increase in zone granularity allows to i) increase acc and ii) decrease redu. Increasing the number of zones within a certain area means there is a higher chance that mobile devices will exit one zone and enter another, hence data from the device becomes less likely to be considered for deduplication.

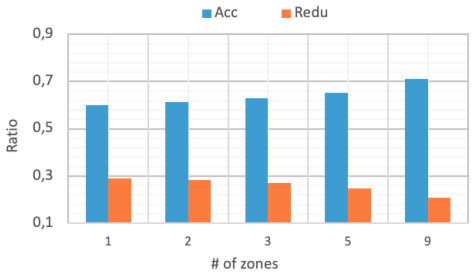
We notice a similar behavior in Figure 5.7 which shows the results of deduplication quality metrics considering sink device data from dataset 3. Here, we highlight the following observations:

- Considering deduplication at edge-only: results show that an increase in zone granularity does not have any impact on the deduplication metrics, since all edge devices in dataset 3 are static devices, and no additional deduplication is performed at the sink level.

- Considering deduplication at sink-only and at edge-and-sink: an increase in zone granularity produces: (i) an increase in *acc* and (ii) a decrease in *redu*. This is because a higher number of zones means less sensors will be collecting data from the same zone in a certain time span. Similarly, $|data_{Trans}|$ from the edges to the sink and $|data_{Stored}|$ at the sink will increase since less data are being deduplicated at the edge level.
- In case only one sensor is collecting data in each zone (considering 9 different zones in our empirical use case), all three deduplications (edge-only, sink-only, and edge-and-sink) will produce the same results since edge devices are not clustered together at the sink-level.

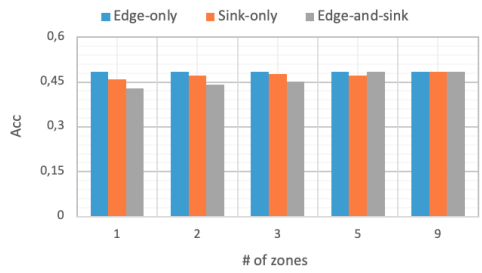


(A) Results with static device data from dataset 1

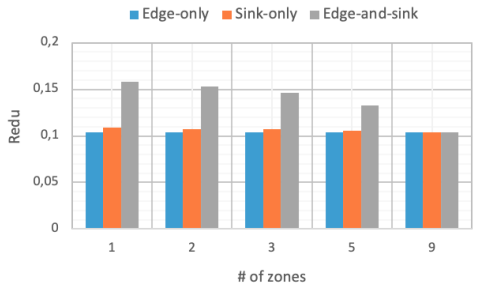


(B) Results with mobile device data from dataset 2

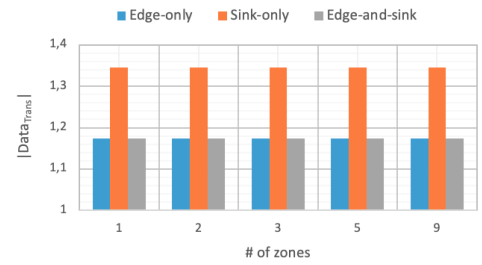
FIGURE 5.6: *acc* and *redu* results with varying zone granularities considering static edge nodes



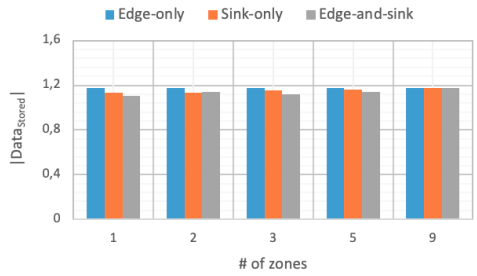
(A) Deduplication accuracy *acc*



(B) Data reduction ratio *redu*



(C) Transmitted data size to sink ($|data_{Trans}|$)

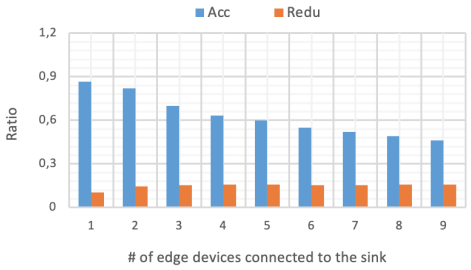


(D) Stored data size at sink ($|data_{Stored}|$)

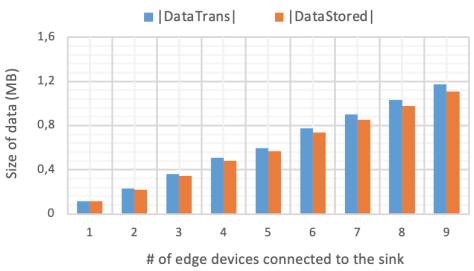
FIGURE 5.7: Deduplication quality metrics obtained with varying zone granularities considering mobile edge nodes

Number of Edge Devices Connected to Sink Nodes In this experiment, we evaluate the impact of changing the number of edge devices connected to the sink node. We consider a 1-zone (1-sink) granularity scenario, and we vary the number of edge

devices in the zone from 1-to-9. We perform this experiment considering deduplication at the edge-and-sink level considering a varying number of edge devices per sink node based on the domain expert and application needs (cf. Challenge 3 - Chapter 1). Figure 5.8 shows deduplication quality metrics applied on sink node data from dataset 3 while varying the number of devices dataset 3 is collected from. Results in Figure 5.8a show a decrease in *acc* and a slight increase *redu* at the sink node, when the number of edge devices is increased per sink node. This is because more data from more sensors is processed at the sink node, where sensors are more dispersed in the zone and might produce different measurements which are not always suitable for deduplication. In addition, results in Figure 5.8b show drastic increases in both $|data_{Trans}|$ and $|data_{Stored}|$ when the number of edge devices increases per sink node, highlighting the fact that more data is being transmitted to and stored at the sink.



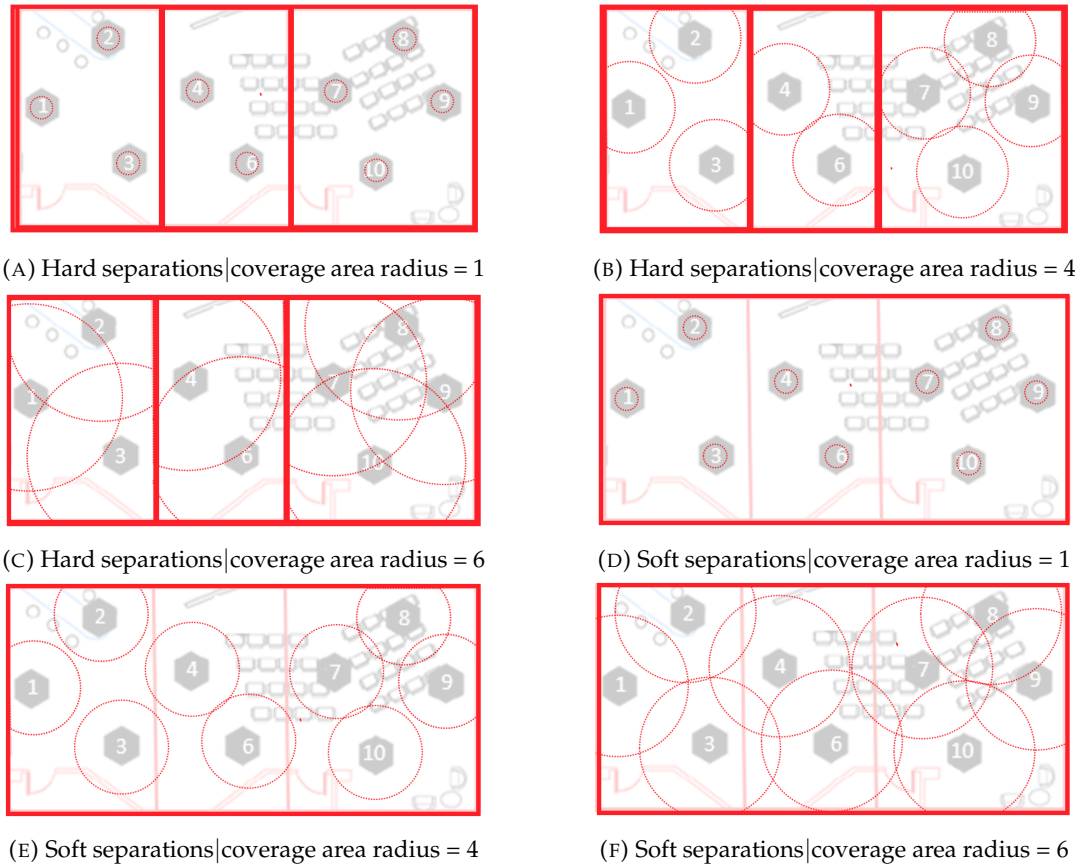
(A) Deduplication accuracy (*acc*) and data reduction ratio (*redu*)



(B) Size of Transmitted & Stored data ($|data_{Trans}|$) & ($|data_{Stored}|$) at the sink

FIGURE 5.8: Deduplication quality metrics applied on sink node data (dataset 3), when varying the number of devices per sink zone

Sensor Coverage Area Size Evaluation In this experiment, we evaluate the impact of varying the radius of sensor coverage areas on the deduplication process. We consider in Figure 5.9 multiple configurations of sink coverage areas divided into three zones including both hard and soft separations following our use case scenarios (cf. Section 5.1). For every use case, we vary the sensor coverage area radius from 1 (i.e., data is only collected at the exact location of the sensor) to whole sink zone (i.e., data is collected from the whole sink zone where the sensor device is located). Deduplication is performed at the edge-and-sink level.



(A) Hard separations|coverage area radius = 1

(B) Hard separations|coverage area radius = 4

(C) Hard separations|coverage area radius = 6

(D) Soft separations|coverage area radius = 1

(E) Soft separations|coverage area radius = 4

(F) Soft separations|coverage area radius = 6

FIGURE 5.9: Sample sink zone use cases with sensor coverage area radius variations using an extract of the Micra2Dot dataset

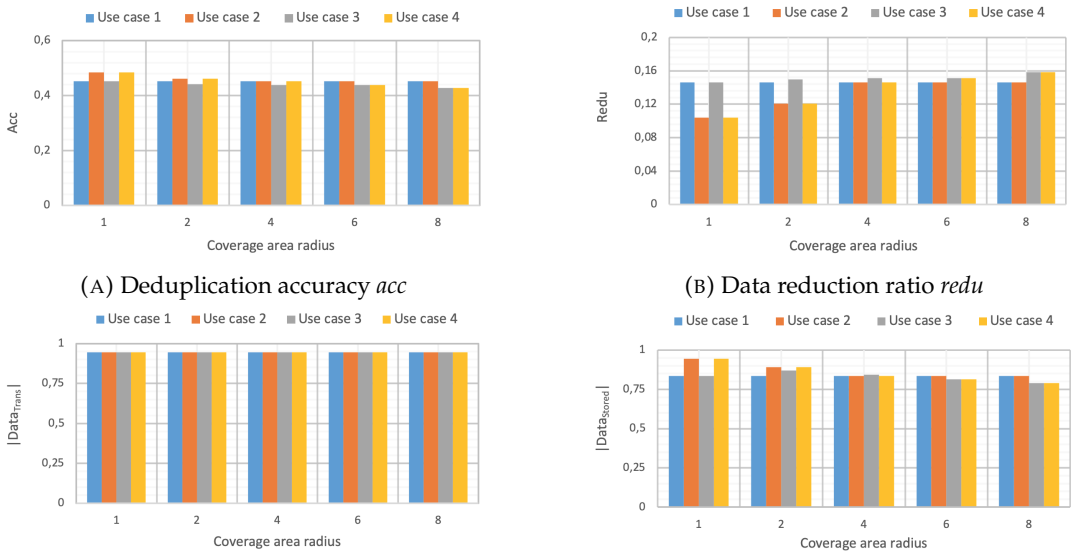
Figure 5.10 shows deduplication quality metrics when applied on sink node data from dataset 3. Here, we highlight the following observations:

- In case 1 – zone-based with hard separations (Figure 5.10a): *acc*, *redu*, and $|data_{Stored}|$ are not affected by the size of the sensor's coverage area, since the deduplication process is only affected by the location of the sensor, and not its coverage area. Note that $|data_{Trans}|$ in unaffected in all cases (cf. Figure 5.10c) since the present use case variations do not target the data transmitted from edge to sink: deduplication at the edge level is not affected by a sensor's coverage area.

- In case 2 – zone-and-coverage based with hard separations (Figure 5.10a) and in case 4 – zone-and-coverage based with soft separations (cf. Figure 5.10b), results show that an increase in the coverage area size leads to (i) a decrease in *acc*; (ii) an increase *redu*; and (iii) a decrease in $|data_{Stored}|$, since more sensors are being considered for deduplication (clustered together) due to their coverage area overlaps (cf. Figure 5.10d). However, we notice that *acc* and *redu* for case 2 stop decreasing/increasing and stabilize after reaching the limit of the zone area, since the zone separations in this case are hard (i.e., the number of sensors that could be clustered together will reach its limit).

- In case 3 – zone-based with soft separations (Figure 5.10b), *redu* increases with the increase in coverage area size since separations between zones are soft, since more sensors in this case will belong to more than one zone at the same time.

- For use cases 2 and 4 (zone-and-coverage), *redu* levels are smaller than those for cases 1 and 3 (zone-based only). This is because cases 2 and 4 consider the similarities between sensor coverage areas belonging to the same zones, allowing to detect redundant measurements between similar sensors, thus affecting the deduplication process accordingly.
- For a coverage area radius =1, *acc* and *redu* accuracy for both cases 1 and 2 are the same. This is because the coverage area size is too small, and hence, each sensor will belong to one zone whether the separations are hard or soft. $|data_{Stored}|$ increases with *redu* since deduplication is performed on the edge-and-sink level.



(A) Deduplication accuracy *acc*

(B) Data reduction ratio *redu*

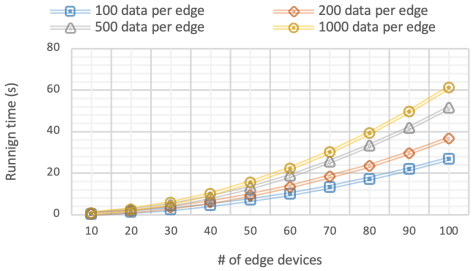
(C) Transmitted data size to sink ($|data_{Trans}|$)

(D) Stored data size at sink ($|data_{Stored}|$)

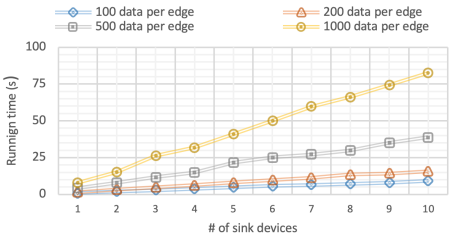
FIGURE 5.10: Deduplication quality metrics applied on sink node data (dataset 3), when varying sensor coverage area radius with multiple hard/soft separation use cases

Performance Evaluation

We evaluate here the efficiency in terms of execution time. Results in Figure 5.11a highlight the polynomial complexity of FREDD's edge-and-sink deduplication process when varying the number of edge devices per sink node, reflecting $\mathcal{O}(E^2)$ time. Figure 5.11b highlights FREDD's linear complexity when varying the number of sink nodes, reflecting $\mathcal{O}(E \times S)$ time.



(A) Edge-and-sink level processing time when varying the number of edge devices per sink note



(B) Edge-and-sink level processing time when varying the number of sink nodes, considering a fixed # of 10 edge devices per sink

FIGURE 5.11: FREDD's time performance consider edge-and-sink level deduplication processes with varying parameters

Baseline Comparison with Existing Approaches

In order to further evaluate our solution, we conducted a comparative study to assess its effectiveness with respect to recent alternatives in the literature. On the one hand, our solution (i) handles redundancies at both edge device and sink device levels of the network (cf. Challenges 1 and 2 of our motivation scenario illustrated in Chapter 1); (ii) handles static and mobile devices, taking into account zone separations (hard/soft) and coverage area variations (Challenge 2); and (iii) allows adapting the deduplication process behavior following the expert's needs (Challenge 3). Most of the latter challenges are overlooked by existing solutions, except for device mobility which is handled in [24] (cf. comparative Table 5.2). On the other hand, our solution performs fuzzy processing, allowing for improved deduplication quality compared with the crisp deduplication processes employed in existing solutions.

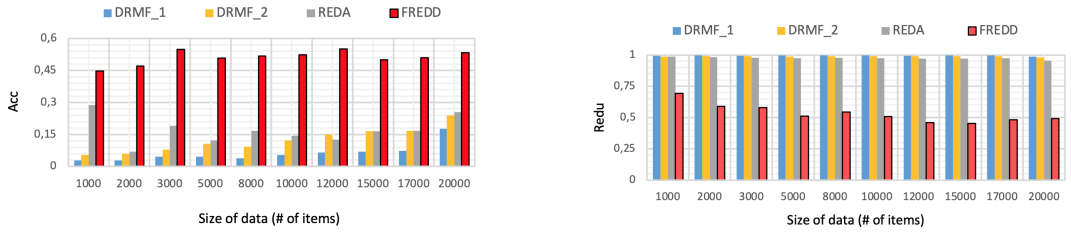
TABLE 5.2: Comparing FREDD with alternative solutions

	Challenge 1		Challenge 2			Challenge 3
	Edge Level Deduplication (Static)	Edge Level Deduplication (Mobile)	Sink Level Deduplication	Zone Separations (Hard/Soft)	Sensor Coverage Area Size	Domain Expert Control
SVM [50]	✗	✗	✓	✗	✗	✗
CWCA [24]	✗	✗	✓	✗	✗	✓
REDA [53]	✓	✗	✗	✗	✗	✗
DRMF [20]	✓	✓	✗	✗	✗	✗
FREDD	✓	✓	✓	✓	✓	✓

We experimentally compare our method's effectiveness with two of its most recent alternatives: i.e., REDA [53] and DRMF [20]. To test REDA, we consider the crisp humidity ranges shown in Figure 4.11a. To test FREDD, we consider the fuzzy humidity ranges in Figure 4.11b where 11 pattern codes are defined, and we set the deduplication threshold to 0.8. We also consider two variations of DRMF: (i) the first one with a deviation threshold equal to one quarter of the width of the crisp range $\delta = 3/4$ (which we refer to as DRMF_1); and (ii) the second one with a deviation threshold equal to one eighth of the width of the crisp range $\delta = 3/8$ (which we refer to as DRMF_2).

Figure 5.12 shows the *acc* and *redu* results obtained from each of the four algorithms when varying the number of data measurements of dataset1. Results show that FREDD consistently achieves the best *acc* results across all data variations compared with both REDA, DRMF_1, and DRMF_2. This is specifically due to FREDD's

fuzzy processing capability, allowing to detect approximate redundancies and process them for deduplication, compared with the crisp decision-making processes performed by both REDA and DRMF.



(A) Deduplication accuracy (*acc*)

(B) Data reduction ratio (*redu*)

FIGURE 5.12: Comparison of the deduplication quality metrics between RED, DRMF_1, DRMF_2, and FREDD, when varying the number of data measurements of dataset1

To further explain the results in Figure 5.12, we conduct a second experiment where we compare the decision-making behavior of each algorithm applied on different pairs of humidity data measurement; the first data item is fixed at a certain value, while the second item is varied within a controlled range. Figure 5.13 shows the percentage of deduplication produced by each algorithm for a first humidity value of $39.5\mu g/m^3$, and the second value with a variation range of $\pm 2.5\mu g/m^3$.

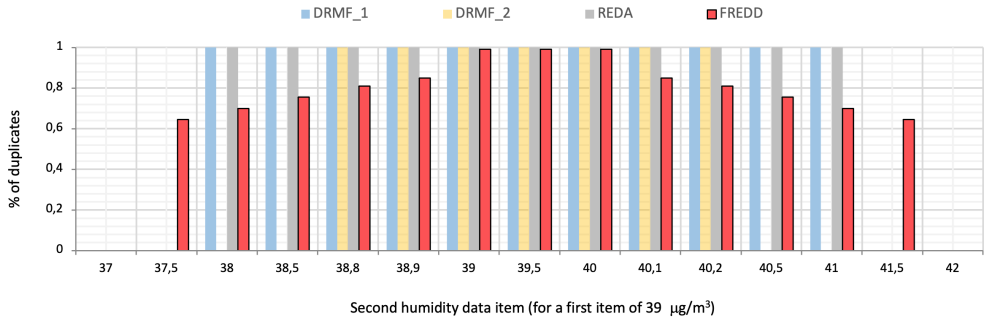


FIGURE 5.13: Percentage of deduplicates when fixing the first humidity data to $39.5\mu g/m^3$ and varying the second between $[37, 42]\mu g/m^3$

Considering REDA's results, all values that lie between $[38, 41]\mu g/m^3$ are considered automatic duplicates (i.e., 100% duplicates) and are assigned the same pattern code. With DRMF_1, considering a delta $\delta = 2/3$ and given a cluster centroid of $39.5\mu g/m^3$, all values that lie between $[38, 41]\mu g/m^3$ are considered duplicates, and a new cluster centroid is computed outside those boundaries. Such a behavior decreases *acc* since a good number of pairs are considered automatic duplicates, producing 100% duplicates in Figure 5.13. A similar behavior is also noticed for DRMF_2. In contrast, each pattern code range in FREDD is divided into: (i) a crisp range where pairs are automatically considered duplicates (i.e., from $[39, 40]\mu g/m^3$); and (ii) a fuzzy range (i.e., between $[37, 39]\mu g/m^3$ and $[40, 42]\mu g/m^3$) where boundaries from different other ranges overlap. In the fuzzy range, the deduplication decision is made based on a fuzzy inference system and a set of fuzzy rules, allowing the percentage of duplicates to vary accordingly (e.g., for a second value of $38\mu g/m^3$, the percentage of duplicates is 70%). By deciding on an appropriate deduplication threshold (e.g., 0.8), only pairs that result in a deduplication percentage bigger than 80% will be considered duplicates. Implementing

such behavior makes the deduplication decision more accurate and intelligent. Less duplicate pairs are considered automatic duplicates and the accuracy of the deduplication process increases accordingly (as shown in Figure 5.13).

We have also compared FREDD's time complexity with its recent alternatives, REDA, DRMF_1, and DRMF_2, using different configurations when varying the number of data items per edge node. Figure 5.14 shows representative running time results considering a fix data size per edge device = 1000 items and a fixed number of edges per sink node = 10. Results show that REDA is the most efficient approach due to its fast and crisp pattern code assignment approach. FREDD requires more processing time than REDA due to its fuzzy computation process. DRMF is seemingly the most time consuming approach due to its data clustering process which is utilized to perform data aggregation. This means that our approach is able to produce improved deduplication quality while increasing execution time compared with the crisp REDA approach, and outperforming the execution time of DRMF.

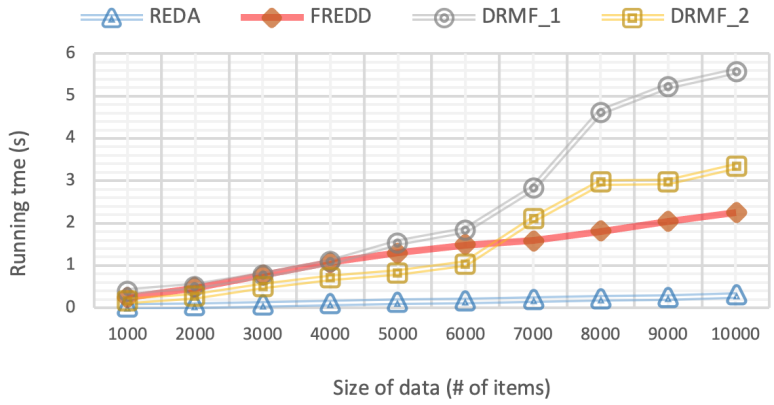


FIGURE 5.14: FREDD's time performance compared with its recent alternatives, considering a fixed data size of 1000 items per edge, and a fixed number of 10 edge devices per sink node

5.5 Recap

In this chapter, we address the need to detect data redundancies at the core of a CE (i.e., at the sink node level). We build upon FREDD: the Fuzzy Redundancy Elimination for Data Deduplication approach introduced in Chapter 4 and consider various environmental/physical, sensor-related, spatial features to detect redundancies on the sink level. This allows us to cover data redundancy detection in a more global manner in connected environments. This entails covering multiple scenarios: (i) considering both static and mobile edge devices (cf. Challenge 1 in our motivating scenario); (ii) considering sink level deduplication by taking into account hard-separated and soft-separated zones, and different sensor coverage areas (cf. Challenge 2); and (iii) allowing the user or a domain expert to adapt and configure the redundancy detection process (cf. Challenge 3). Various experiments on a real-world dataset highlight FREDD's potential and improvement compared with existing solutions. Having extensively covered data redundancy detection in Chapters 3-5, we shift our attention in the following chapter to redundancy removal/cleaning (i.e., the deduplication part of the overall DRMCE process illustrated in Figure 1.5 - Chapter 1).

Chapter 6

Data Redundancy Cleaning

The previous chapters thoroughly discussed the process of detecting redundancies in a CE (on the edge and sink levels). Our proposals considered the following redundancy detection aspects: (i) considering both static and mobile edge devices (cf. Challenge 1 of our motivating scenario - Chapter 1); (ii) considering physical, spatial, and device-related features to enable a more contextually-aware data redundancy detection at the sink level (cf. Challenge 2 of our motivating scenario - Chapter 1); and (iii) considering the user needs/specifications when configuring the redundancy detection process in order to allow domain experts to better detect redundancies in their connected environments (cf. Challenge 3 of our motivating scenario - Chapter 1).

In this work, we focus on cleaning data redundancies in CEs. Existing approaches often disregard data consumer needs/requirements when cleaning the data redundancies. Most of the existing approaches summarize a set of redundant data to reduce it to one significant representative data item. This is traditionally done using various functions (e.g., replacing the set by the mean, or median value). In this proposal, we provide users/domain experts with more flexibility when removing duplicates from their data. We provide for data consumers a mean for defining their requirements and inject the latter in the deduplication process (at the redundancy cleaning step). This enables the configuration and customization of this process in order to adapt to consumers' requirements (cf. Challenge 3 of our motivating scenario - Chapter 1).

To do so, we cover in this chapter the data redundancy cleaning step of the DRMCE process (proposed in Chapter 1) and present two data cleaning modes: (i) the auto-clean mode for conventional and fast data cleaning (to be used if the users don't have specific needs for the deduplication step); and (ii) the consumer-centric cleaning mode that consider various data consumer needs (e.g., domain expert, device, database, data processing service needs) and adapts the cleaning process accordingly. We describe how this is used with DRMF and FREDD in the DRMCE global process. We also provide some experiments and results from this ongoing work.

6.1 Introduction

As previously mentioned, the data collected from CEs often suffers from various inconsistencies such as redundancies, anomalies, and missing values [26, 55]. Therefore, data refining and pre-processing are needed in order to prepare the data for advanced processing. This enables the exploitation of useful, concise, and complete data in decision making systems that aid users in handling their connected environments. In the case of our study on data redundancy management, identifying and cleaning unnecessary data (i.e., duplicates) is beneficial for (i) querying data efficiently from edge devices and/or the network's database; (ii) querying spatial-temporal data efficiently from mobile edge devices; (iii) conserving the limited resources of edge devices (e.g., battery, memory, bandwidth); (iv) conserving the network's shared resources (e.g., central database, bandwidth); and (v) providing ready-to-use data collections for consumers based on their needs/requirements.

Since we already extensively detailed data redundancy identification/detection in the previous chapters, we focus here on the second part of the deduplciation process (i.e., the redundancy cleaning/removal/elimination). Existing works [34, 47, 48] (cf. Chapter 2) have targeted data redundancy elimination in CEs, however most of them provide automated, quick-clean solutions that eliminate or summarize redundancies. Their cleaning mechanisms are applied similarly for every set of redundant data without considering the differing needs of the data consumers that require the data. This is important because in a CE, different data consumers (e.g., a human user retrieving data, a database storing data, devices exchanging data, and services processing data), might have different deduplication needs or requirements. To illustrate this, we provide some examples here:

1. A user might need the most concise answer for his/her data retrieval query. This affects the deduplication process since it impacts the deduplication ratio to be used when cleaning the identified redundant data.

2. A database administrator might specify a quota for data storage per device (to improve its data storage strategy and conserve its resources). This affects the deduplication process since it impacts the deduplication ratio when cleaning the identified redundant data to provide an output data size does not surpass the threshold/quota required by the database.

3. A device might need to exchange data with other devices on the edge of the network (where resources are often limited). In this scenario, we consider devices to be prosumers (i.e., producers when generating data and consumers when asking for data). In this case, a device might require data exchange that does not overload its processor, or consumer too much battery. This is dictated by the amount of transmitted/recieved data which also depends on the deduplication ratio applied when removing the identified data redundancies prior to inter-device data exchange.

4. A data statistics service might require the data with the redundancies in order to avoid biasing the calculated averages and statistics. However, a machine learning service might require the data to be fully deduplicated to avoid excessive training of unnecessary data which could prove costly.

In order to address the aforementioned needs, we provide a consumer-centric data redundancy removal process that takes into account the various needs of

different data consumers and considers them in the deduplication process (cf. Challenge 3 of our motivating scenario - Chapter 1). We discuss here how the requirements are defined and used. Moreover, we illustrate how they can be used to adapt the deduplication process to data consumer needs/requirements. In addition, we also provide the automated cleaning mode (via summarizing redundant sets of data based on the mean, or average value) in the DRMCE process if there are no specific deduplication requirements to be considered. We also show how this proposal can be coupled with DRMF and FREDD. Finally, we present some preliminary experiments and results.

The remainder of this chapter is organized as follows. Section 6.2 provides some preliminaries on how the data consumer requirements are defined and represented/used in DRMCE. Section 6.3 provides an overview of how the consumer-centric deduplication process is coupled with DRMF and FREDD. Then, Section 6.4 discusses some preliminary experiments and results. Finally, Section 6.5 recaps this study.

6.2 Preliminaries & Illustration Example

Once redundancies are identified, the redundancy removal process occurs. Here, we propose two redundancy removal modes: (i) the auto-removal mode summarizes a sequence of redundancies into one representative data item using the median or mean representative values; and (ii) the consumer-centric mode considers a data consumer request that describes the deduplication requirements/conditions when removing redundancies. Following our running example data from Tables 4.11 and A.1, the identified humidity and temperature redundancies can be removed using the auto-removal median function as illustrated in Tables 6.1 and 6.2 respectively.

TABLE 6.1: Auto-Clean Mode (via Median) - Humidity Data Collection

Attribute	Value	Temporal Stamp		Location Stamp		Source
		Format	Value	Format	Value	
Humidity	92 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	Cartesian	(1, 2, 3)	S1
Humidity	103 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	(5, 2, 7)	S1

TABLE 6.2: Auto-Clean Mode (via Median) - Temperature Data Collection

Attribute	Value	Temporal Stamp		Location Stamp		Source
		Format	Value	Format	Value	
Temperature	16° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	(2, 5, 3)	S2
Temperature	21° C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	(8, 6, 3)	S2

Note that data consumers might have different needs for redundancy removal. For instance, a database could require a specific amount of data from each device per day (thus affecting the deduplication ratio). An expert could have different requirements based on available resources (e.g., high deduplication ratio if resources are low). Similarly, devices and services consuming data might have specifications for the redundancy removal process. In order to consider these needs (cf. Challenge

3), data consumers can provide their requirements in the form of simple consumer requests that the module translates into redundancy removal rules:

Definition 8 (Data Consumer Request)**.** *We define a consumer request req as a 3-tuple:*

$$req = \langle c_{id}, s_{id}, P \rangle \quad where: \tag{6.1}$$

- c_{id} *is the data consumer identifier*
- s_{id} *is the data source (device) identifier*
- P *is a set of consumer preferences for the sensory data produced by the data source* ■

Definition 9 (Data Consumer Preferences)**.** *Consumer preferences* $p \in P$ *can be expressed as a 4-tuple:*

$$p = \langle target_a, freq, type, f_{rep} \rangle \quad where: \tag{6.2}$$

- $target_a$ *is the data measurement targeted by the request (e.g., humidity, temperature)*
- $freq$ *is the data consumption frequency (expressed in units of time, e.g., per second, every 30 seconds, every hour)*
- $type$ *is the deduplication type (expressed in terms of required deduplication ratio or percentage, or allowed memory size, CPU consumption, or energy consumption levels during deduplication)*
- f_{rep} *is the data item representative selection function (including mean, median, minimum, maximum, as well as earliest value and latest value based on the data item's time stamp)* ■

To illustrate the consumer-centric deduplication mode, consider the same humidity data collection depicted in Table 4.11. In the following, we will define an instance of a data consumer request (following Defintion 8) having one instance of data consumer preference (following Definition 9) and transform this request into a deduplication rule (cf. Figure 6.1) and apply it to generate the redundancy-free result (cf. Table 6.3):

Step 1: Consumer Request & Preference: Consider the following request by user 1 that would like to query data from device 1 using the following data consumer request:

$$req = \langle user_1,\ device_1,\ p_1 \rangle$$

The user expresses his preference of querying humidity data every day with a deduplication ratio of 50% using the mean representative function via the following preference $p1$:

$$p_1 = \langle humidity,\ everyDay,\ ratio = 50\%,\ mean \rangle$$

The user's request/preference are tranlated into the following rule:

```
<req_1>
    <c_id>user1</c_id>
    <d_id>device1</d_id>
    <P>
        <p>
            <target_a>Humidity</target_a>
            <freq>everyDay</freq>
            <type>ratio = 50%</type>
            <f_rep>Mean</f_rep>

        </p>
    </P>
</req_1>
```

FIGURE 6.1: Sample User Request Rule

Finally, the deduplication result is shown in the following table:

TABLE 6.3: Consumer-Centric Mode (via req_1/p_1) - Humidity Data Collection

Attribute	Value	Temporal Stamp		Location Stamp		Source
		Format	Value	Format	Value	
Humidity	93 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	(2.5, 2, 5)	S1
Humidity	103.5 $\mu g/m^3$	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:30	Cartesian	(6, 2, 7)	S1

6.3 Linking Deduplication to DRMF & FREDD

The DRMCE process (cf. Figure 1.5) provides options for data redundancy detection (i.e., DRMF for edge level data deduplication, and FREDD for edge-and-sink data deduplication). Then, regardless of the chosen technique for redundancy identification a set of redundant data is forwarded to the cleaning process. At this stage, the user also gets to choose if he/she would like to apply the auto-clean mode or the consumer-centric mode. Figure 6.2 shows a non detailed overview of how the redundancy detection algorithms link to the redundancy cleaning modules.

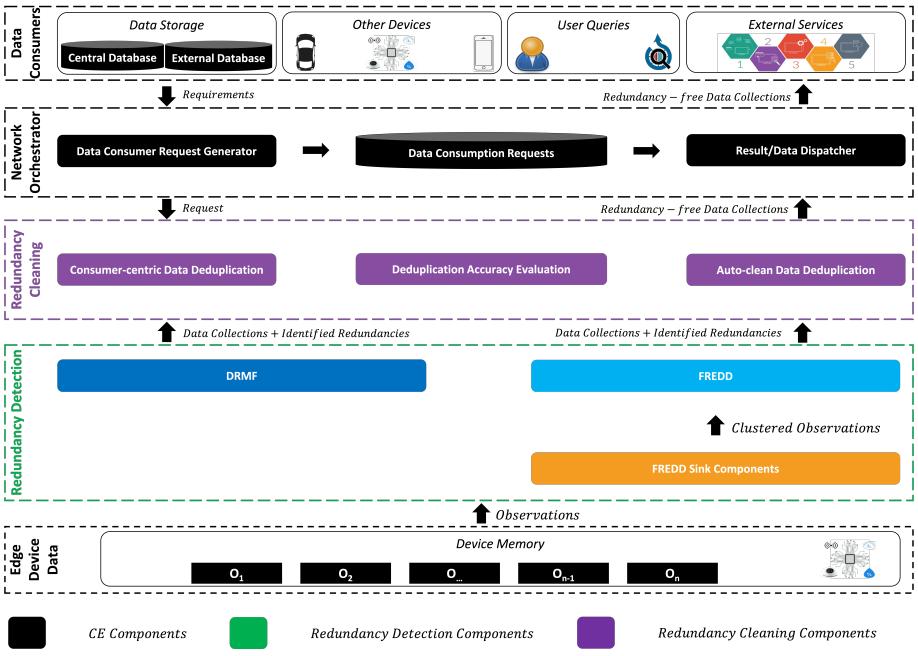


FIGURE 6.2: Linking Redundancy Detection & Cleaning - Overview

6.3.1 Redundancy Cleaning Components

Figure 6.3 shows a detailed overview of how the redundancy detection algorithms (DRMF - previously detailed in Chapter 3, and FREDD - previously detailed in Chapters 4-5) link with the different mechanisms for redundancy removal. We note that when redundancies are identified (via DRMF or FREDD), one can either clean via the automated mode or the consumer-centric mode. When using the auto-clean mode, the *Auto-clean Data Deduplication* module is triggered. However, when considering data consumer needs, the *Consumer-centric Data Deduplication* module retrieves the data consumer request from a central repository and cleans the duplicates accordingly. We also mention that the *Deduplication Accuracy Evaluation* module evaluates the accuracy of the deduplication process and compares it to an acceptable system-defined threshold. If the accuracy is not good enough, one could adjust the redundancy dtection configuration in order to detect redundancies more accurately (e.g., trigger the detection tuning core of DRMF to automatically re-adjust the parameters of the redundancy detection algorithms, or one can change the configuration of the fuzzy engine in FREDD to achieve better results). Finally, redundancy-free data is sent to the data consumer.

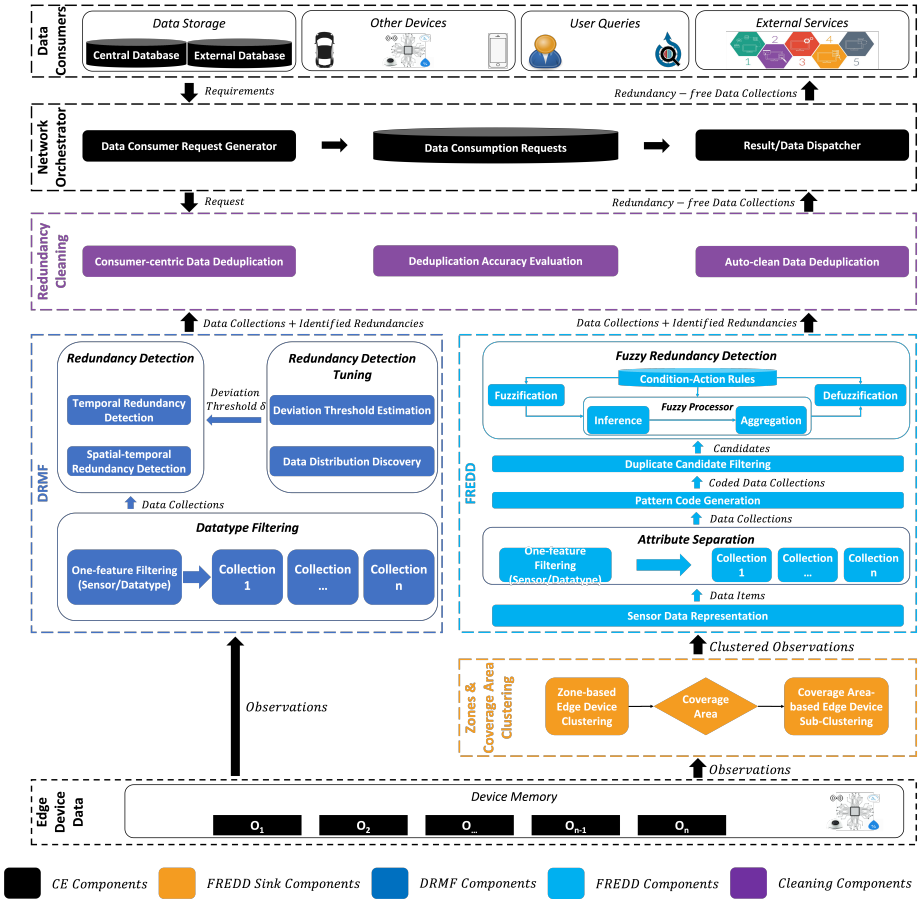


FIGURE 6.3: Linking Redundancy Detection & Cleaning - Detailed

6.4 Implementation & Preliminary Evaluation

As previously mentioned, the consumer-centric cleaner is still an ongoing work. The implementation of the request to rule parser is not yet finished. Therefore, we cannot extensively evaluate the cleaner's performance and accuracy here. However, in order to show some preliminary results, we hard-coded two data cleaning rules in a prototype developed in Python to show the feasibility of the proposal. We experimented with the same Intel Lab datasets described in the previous chapters and deduplicated sets of identified redundancies using the following two rules (cf. Figures 6.4a and 6.4b):

```
<req_1>
    <c_id>user1</c_id>
    <d_id>device1</d_id>
    <P>
        <p>
            <target_a>Humidity</target_a>
            <freq>everyDay</freq>
            <type>ratio = 40%</type>
            <f_rep>Mean</f_rep>
        </p>
    </P>
</req_1>
```

(A) Consumer request 1

```
<req_2>
    <c_id>user2</c_id>
    <d_id>device2</d_id>
    <P>
        <p>
            <target_a>Humidity</target_a>
            <freq>everyDay</freq>
            <type>ratio = 60%</type>
            <f_rep>Mean</f_rep>
        </p>
    </P>
</req_2>
```

(B) Consumer request 2

Consumer Requests 1 and 2 are ratio-based. This means that the user specifically defines a deduplication ratio to be applied when cleaning duplicate values identified by the redundancy detection algorithm. Figures 6.5 and 6.6 show the results of deduplicating using consumer requests 1 and 2 respectively (for these experiments we identified redundancies on the entire humidity dataset using DRMF).

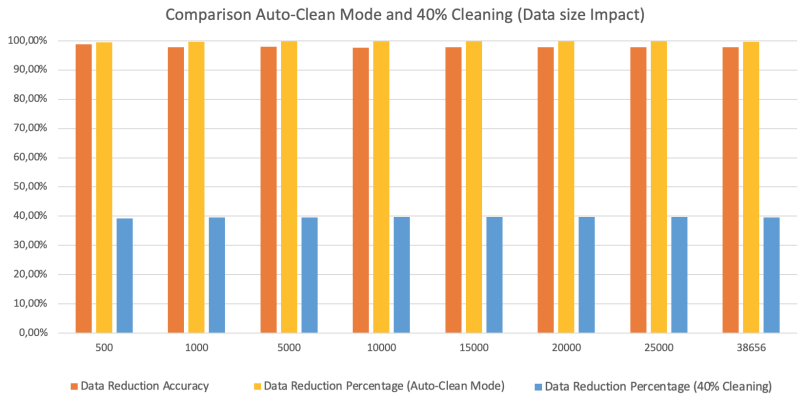


FIGURE 6.5: Deduplication Result - Consumer Request 1 .vs. Auto-Clean

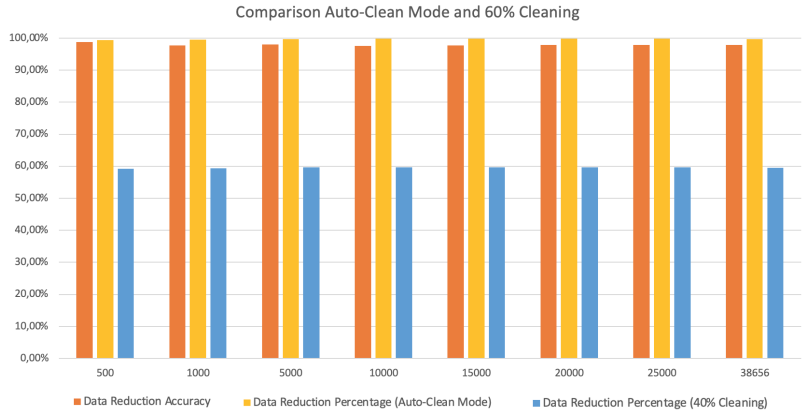


FIGURE 6.6: Deduplication Result - Consumer Request 2 .vs. Auto-Clean

6.5 Recap

Redundancy management consists of two main steps: (i) redundancy identification; (ii) redundancy cleaning. We previously covered redundancy identification in Chapters 3-5. In this chapter, we focus on redundancy cleaning by proposing two different mechanisms in order to remove redundancies once identified. We complete our DRMCE pipeline with an automated cleaning mode (auto-clean mode) that handles redundancies like most of the existing approaches in the literature, and a novel data consumer-centric redundancy cleaning mode that considers various data consumers (e.g., users, databases, devices, and services) and their requirements when cleaning duplicates. Although this is still an ongoing work, we show how these two modes can be used with the previously introduced redundancy detection methods (i.e., DRMF and FREDD). Moreover, we illustrate how the two cleaning modes work and provide some preliminary experiments and results that highlight the potential of the consumer-centric mode. A more complete implementation and evaluation of this study will be provided in a dedicated work.

Chapter 7

Conclusion & Future Works

7.1 Report Recap

7.1.1 In Chapter 1

We introduce the main motives behind the interest in connected environments and how the latter are defined and organized. Then, we shift our attention towards interesting data management topics such as anomaly detection/correction, missing data management, and data deduplication. Thereafter, we present a smart hospital connected environment to highlight are motivations in regards to data redundancy management in connected environments. This scenario allows us to identify three main challenges: (i) detecting redundancies at the edge of the network on static and mobile devices; (ii) detecting redundancies at the core of the network by considering different environmental, physical, and sensor-related features; and (iii) allowing users (e.g., domain experts) more flexibility when configuring the redundancy detection and cleaning processes. To do so, we introduce the DRMCE: Data Redundancy Management in Connected Environment process/pipeline to identify and remove data redundancies in various connected environments. Finally, this chapter lists the thesis's objectives and states the report's structure.

7.1.2 In Chapter 2

We review the literature on redundancy identification and redundancy removal approaches. We provide summaries of each cited work and propose a set of criteria to compare each category of approaches. Finally, we conduct a comparative study to highlight the needs behind the proposals of this thesis.

7.1.3 In Chapter 3

We propose DRMF: the Data Redundancy Management for leaF-edge. This proposal addresses Challenge 1 by identifying data redundancies at the source (i.e., on edge sensing devices). Moreover, we define two types of redundancies here: (i) temporal redundancies to be detected on static edge devices; and (ii) spatial-temporal redundancies to be detected on mobile edge devices. We formally define the terminology used in this work. Then, we provide a detailed description of the different modules of DRMF. We evaluate our proposal's performance and accuracy hen detecting redundancies before recapping the chapter with a discussion of the main limitation of this work: crisp threshold usage during the clustering might lead to missing some data redundancies or flagging non-redundant data as redundancies.

7.1.4 In Chapter 4

We address the limitations of DRMF by proposing an improved approach for data redundancy detection at the edge of the network (also addressing Challenge 1). To do so, we propose the FREDD method: Fuzzy Redundancy Elimination for Data Deduplication, which uses fuzzy reasoning to overcome the crisp threshold limitation. This proposal also considers static and mobile edge devices (cf. Challenge 1) and provides domain experts with the flexibility needed to customize/configure the redundancy identification process in order to adapt it to their needs/requirements. We provide a preliminary on fuzzy reasoning, detail the FREDD architecture and explain its modules while providing an illustration use case. Finally, we extensively evaluate the performance and accuracy of FREDD before concluding the chapter.

7.1.5 In Chapter 5

We shift our attention towards detecting redundancies in the core of the network (i.e., at the sink level). We identify key environmental, physical, and sensor-related features to be considered (e.g., spatial setup of the environment, zoning, hard/soft zone separations, sensor coverage areas) and extend FREDD to be used for data deduplication at the sink. This study mainly addresses Challenge 2 in order to provide data deduplication at different layers of connected environments. We detail the impact of the added constraints and how FREDD can consider them in order to achieve deduplication at the core. Similarly to the previous chapter, we extensively evaluate this proposal and provide a comparative analysis with existing works before concluding the chapter.

7.1.6 In Chapter 6

We focus on redundancy cleaning by proposing two modes for deduplication once redundancies are identified. We implement first the auto-clean mode (heavily used in the literature), that reduces a set of redundant data to one representative data item (e.g. the mean, or median value). Then, we propose a data consumer-centric mode that considers different data consumer needs/requirements (cf. Challenge 3) and transforms them into deduplication rules that affect the cleaning phase in order to produce results according to data consumer needs. Although this proposal is still an ongoing work, we provide an overview of its framework and how it links to our redundancy detection proposals in DRMCE (i.e., DRMF and FREDD). Then, we show a couple of preliminary experiments and results before concluding the chapter.

7.2 Future Research Directions

7.2.1 DRMF

We plan to improve the deviation threshold estimation by considering pattern recognition and providing multiple thresholds for a data collection. The current algorithm tuning modules use conventional methods that generate one threshold for an entire subset of data. Although this might be good for data with low fluctuations, if the data is dynamic and varies a lot then multiple thresholds per data sections based on detected patterns could prove more accurate.

7.2.2 FREDD

We are currently investigating the use of parametric learners [69] and meta-heuristic algorithms [70] allowing to (semi) automatically configure the pattern codes' interval ranges and the corresponding fuzzy rules based on expert defined, data related, or application related features.

In the near future, we plan to extend this work to cover data redundancies at the base station level of the network, where data is aggregated from multiple sink nodes. In this context, different cases need to be considered including sink node mobility, sink node coverage area overlapping, and inter-sink collaboration.

We also aim to detect composite redundancies [71] that are generated by data fusion from multiple sensors, where deduplication would be handled at the edge, sink, and base station levels of the network. These entail special challenges depending on the structure, connectivity, dynamics, and overall properties of the connected environment.

In the long run, we plan to investigate data recovery [72, 73] in CEs, including damage assessment and recovery from deduplicated data.

7.2.3 Redundancy Cleaning

We plan to investigate rule markup languages and rule parsers in order to efficiently transform user requests/queries that express deduplication preferences into machine-readable and executable rules.

Also, we aim to continue the ongoing implementation and evaluation work for the consumer-centric deduplication mode.

7.2.4 Other Research Directions

CEs often contain a variety of sensors producing different datatypes. In this thesis, we covered scalar (numeric) data deduplication. However, the aforementioned environments can contain sensors that produce multimedia data (e.g., images, audio/video segments). In addition, one can find smart semantic sensors in these environments that provide textual data instead of the numerical values. This constitutes a need for multimedia and textual data deduplication in connected environments.

Finally, we are currently investigation a CE decision maker that will help users/environment managers decide where to perform data deduplication (e.g., at the edge only, at the sink only, at the end and sink). We find this work interesting since it allows users to better manage resources and data in their environments with the help of a decision maker that will provide specific recommendations based on the current status of the devices, the network, and the data flows.

Appendix A

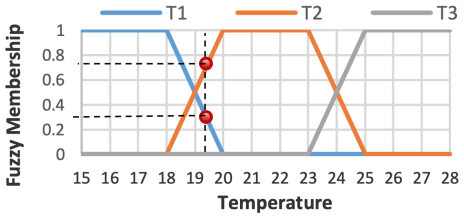
FREDD Computation Example

A.1 Computation Example - Temperature Collection

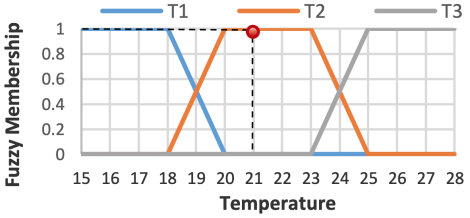
This appendix provides a detailed computation process of FREDD (cf. Chapter 4) for the temperature data collection shown in Table 4.10. For the temperature case, the agent recommends that the inputs 19.5°C and 21°C are 78.2% duplicates which seems accurate following the temperature lookup tables and ranges defined in Table 4.5b (T1 and T2 fuzzy partitions intersect between [18, 20] °C, where 19.5 is closer to the 20°C boundary of T2 than to the 18°C boundary of T1). The fuzzy inference agent produces recommendations that simulate the domain expert's deduplication capability, and behaves following the expert's design choices and needs.

A.1.1 Fuzzification:

Given case 1's input data: $Temp\ DataItem1 = 19.5°C$ and $Temp\ DataItem2 = 21°C$, we compute the corresponding fuzzy membership values (cf. Figure A.1) following the humidity fuzzy functions in Chapter 4 - Figure 4.4b (reported below):



(A) For *Temperature DataItem*1: $fT1(19.5) = 0.25, fT2(19.5) = 0.75, and fT3(19.5) = 0$



(B) For *Temperature DataItem*2:$fT1(21) = 0, fT2(21) = 1, and fT3(21) = 0$

FIGURE A.1: Fuzzification of Temperature *DataItem*1 and *DataItem*2

A.1.2 Condition-Action Rules:

Based on the input membership values, the following condition-action rules are invoked:

Rule 4

```
IF (Temp_DataItem1 is T1) AND (Temp_DataItem2 is T1) THEN
    ↪ DedupStatus is Duplicate
```

Rule 5

```
IF (Temp_DataItem1 is T2) AND (Temp_DataItem2 is T2) THEN
    ↪ DedupStatus is Duplicate
```

A.1.3 Inference:

Applying Mamdani's inference mechanism:

- **Rule 4:** Executing the AND fuzzy operator:
  - $min(f(19.5)_{T1}^{Temperature\_DataItem1}, f(21)_{T1}^{Temperature\_DataItem2}) = min(0.25, 0) = 0$
- Executing the implication fuzzy operator:
  - $f_{Rule4} = min(0, Duplicate(DedupStatus))$

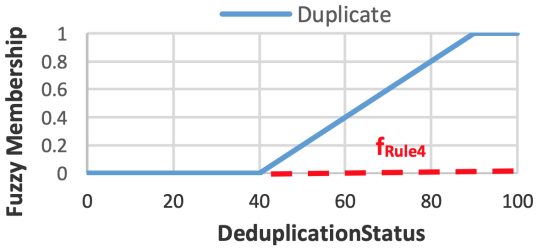


FIGURE A.2: Inference - Rule4

- **Rule 5:** Executing the AND fuzzy operator:
  - $min(f(19.5)_{T2}^{Temperature\_DataItem1}, f(21)_{T2}^{Temperature\_DataItem2}) = min(0.75, 1) = 0.75$
- Executing the implication fuzzy operator:
  - $f_{Rule5} = min(0.75, Duplicate(DedupStatus))$

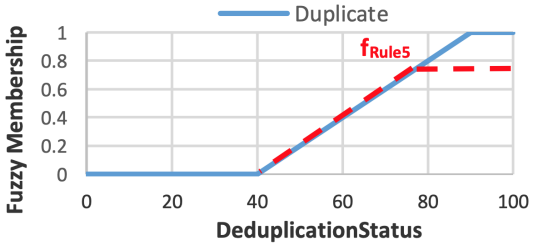


FIGURE A.3: Inference - Rule5

A.1.4 Aggregation:

By applying the maximization aggregation function, $F_{agg} = F_{max} = max(f_{Rule4}, f_{Rule5})$, the agent produces the fuzzy coverage areas subsumed by the inference membership functions (represented in transparent grey color in the below graph).

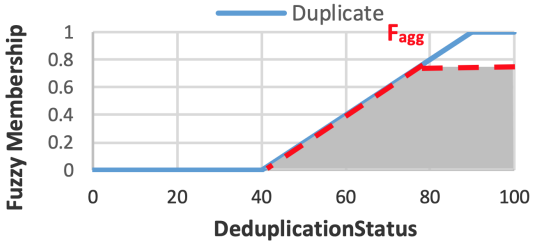


FIGURE A.4: Aggregation

A.1.5 Defuzzification:

The center of gravity defuzzification function is applied on the fuzzy coverage area to compute the corresponding center of gravity point (represented as a red dot in the aggregation graph), and then identify the corresponding deduplication status (on the x axis) the agent's output = 78.2%.

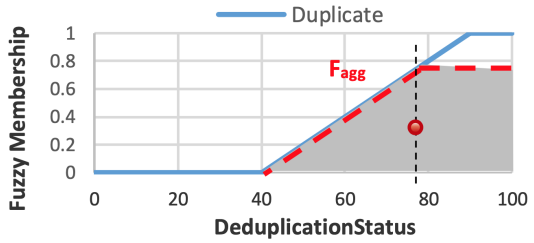


FIGURE A.5: Defuzzification

A.1.6 Result:

Depending on the user or system deduplication threshold, a decision is made whether the two data items are duplicates or not. Given $dedup_{threshold} = 75\%$ in our running example, and since the output of the defuzzification step is 78.2% $\geq dedup_{threshold}$, the agent's final output becomes: $dedupStatus = duplicates$.

Given our running example data from Table 4.10, the identified temporal redundancies following the fuzzy redundancy detection process are shown in Table A.1 where duplicates share the same row color and non duplicates are marked by uncolored rows.

TABLE A.1: Result - Temperature Data Collection

Attribute	Value	Value Pattern Code	Temporal Stamp Format, Value	Location Stamp Format, Value	Zone Pattern Code	Combined Pattern Code	Source
Temperature	16° C	$\{T1\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:01:00	Cartesian (2,5,3)	$\{Z2\}$	$\{T1 - Z2\}$	S2
Temperature	19.5° C	$\{T1, T2\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:02:00	Cartesian (5,6,3)	$\{Z4\}$	$\{T1 - Z4, T2 - Z4\}$	S2
Temperature	21° C	$\{T2\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:03:00	Cartesian (8,6,3)	$\{Z4\}$	$\{T2 - Z4\}$	S2
Temperature	21° C	$\{T2\}$	dd/MM/yyyy hh:mm:ss 10/02/2019 10:05:00	Cartesian (8,6,3)	$\{Z4\}$	$\{T2 - Z4\}$	S2

Bibliography

[1] Ala Al-Fuqaha et al. "Internet of things: A survey on enabling technologies, protocols, and applications". In: *IEEE communications surveys & tutorials* 17.4 (2015), pp. 2347–2376.

[2] John Gantz and David Reinsel. "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east". In: *IDC iView: IDC Analyze the future* 2007.2012 (2012), pp. 1–16.

[3] Dave Evans. "The internet of things: How the next evolution of the internet is changing everything. 2011". In: *URL http://www. cisco. com/web/about/ac79/docs/innov/IoT IBSG 0411FINAL. pdf* (2015).

[4] Stuart Taylor. "The next generation of the Internet revolutionizing the way we work, live, play, and learn". In: *CISCO, San Francisco, CA, USA, CISCO Point of View* 12 (2013), p. 6.

[5] James Manyika et al. *Disruptive technologies: Advances that will transform life, business, and the global economy*. Vol. 180. McKinsey Global Institute San Francisco, CA, 2013.

[6] Nallapaneni Manoj Kumar and Pradeep Kumar Mallick. "The Internet of Things: Insights into the building blocks, component interactions, and architecture layers". In: *Procedia computer science* 132 (2018), pp. 109–117.

[7] Ian F Akyildiz et al. "Wireless sensor networks: a survey". In: *Computer networks* 38.4 (2002), pp. 393–422.

[8] Ahmed Banafa. "IoT and blockchain convergence: benefits and challenges". In: *IEEE Internet of Things* (2017).

[9] Ali Kadhum M Al-Qurabat, Chady Abou Jaoude, and Ali Kadhum Idrees. "Two tier data reduction technique for reducing data transmission in IoT sensors". In: *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE. 2019, pp. 168–173.

[10] Ronald K Pearson. "The problem of disguised missing data". In: *Acm Sigkdd Explorations Newsletter* 8.1 (2006), pp. 83–92.

[11] Kristiaan Pelckmans et al. "Handling missing values in support vector machine classifiers". In: *Neural Networks* 18.5-6 (2005), pp. 684–692.

[12] Irfan Pratama et al. "A review of missing values handling methods on time-series data". In: *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*. IEEE. 2016, pp. 1–6.

[13] Herbert W Marsh. "Pairwise deletion for missing data in structural equation models: Nonpositive definite matrices, parameter estimates, goodness of fit, and adjusted sample sizes". In: *Structural Equation Modeling: A Multidisciplinary Journal* 5.1 (1998), pp. 22–36.

[14] Minale A Abebe et al. "Generic metadata representation framework for social-based event detection, description, and linkage". In: *Knowledge-Based Systems* 188 (2020), p. 104817.

[15] MN Noor et al. *Filling missing data using interpolation methods: Study on the effect of fitting distribution*. Vol. 594. Trans Tech Publ, 2014.

[16] Muhammad Fahim and Alberto Sillitti. "Anomaly detection, analysis and prediction techniques in iot environment: A systematic literature review". In: *IEEE Access* 7 (2019), pp. 81664–81681.

[17] Charu C Aggarwal. "An introduction to outlier analysis". In: *Outlier analysis*. Springer, 2017, pp. 1–34.

[18] Xiuyao Song et al. "Conditional anomaly detection". In: *IEEE Transactions on knowledge and Data Engineering* 19.5 (2007), pp. 631–645.

[19] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. "A survey of anomaly detection techniques in financial domain". In: *Future Generation Computer Systems* 55 (2016), pp. 278–288.

[20] Elio Mansour et al. "Data Redundancy Management in Connected Environments". In: *Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*. 2020, pp. 75–80.

[21] João Paulo and José Pereira. "A survey and classification of storage deduplication systems". In: *ACM Computing Surveys (CSUR)* 47.1 (2014), pp. 1–30.

[22] Jyoti Malhotra and Jagdish Bakal. "A survey and comparative study of data deduplication techniques". In: *2015 International Conference on Pervasive Computing (ICPC)*. IEEE. 2015, pp. 1–5.

[23] Anand Bhalerao and Ambika Pawar. "A survey: On data deduplication for efficiently utilizing cloud storage for big data backups". In: *2017 International Conference on Trends in Electronics and Informatics (ICEI)*. IEEE. 2017, pp. 933–938.

[24] Peter Christen. "A survey of indexing techniques for scalable record linkage and deduplication". In: *IEEE transactions on knowledge and data engineering* 24.9 (2011), pp. 1537–1555.

[25] Srivatsa Maddodi, Girija V Attigeri, and A Kotegar Karunakar. "Data deduplication techniques and analysis". In: *2010 3rd International Conference on Emerging Trends in Engineering and Technology*. IEEE. 2010, pp. 664–668.

[26] Qinlu He, Zhanhuai Li, and Xiao Zhang. "Data deduplication techniques". In: *2010 International Conference on Future Information Technology and Management Engineering*. Vol. 1. IEEE. 2010, pp. 430–433.

[27] Dutch T Meyer and William J Bolosky. "A study of practical deduplication". In: *ACM Transactions on Storage (ToS)* 7.4 (2012), pp. 1–20.

[28] Asif Iqbal Baba et al. "Spatiotemporal data cleansing for indoor RFID tracking data". In: *2013 IEEE 14th International Conference on Mobile Data Management*. Vol. 1. IEEE. 2013, pp. 187–196.

[29] Tuan Dao et al. "Managing redundant content in bandwidth constrained wireless networks". In: *IEEE/ACM Transactions on Networking* 25.2 (2016), pp. 988–1003.

[30] Hassan Harb, Abdallah Makhoul, and Chady Abou Jaoude. "En-route data filtering technique for maximizing wireless sensor network lifetime". In: *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE. 2018, pp. 298–303.

[31] Waleed M Ismael et al. "An In-Networking Double-Layered Data Reduction for Internet of Things (IoT)". In: *Sensors* 19.4 (2019), p. 795.

[32] Yasmin Fathy, Payam Barnaghi, and Rahim Tafazolli. "An adaptive method for data reduction in the internet of things". In: *2018 IEEE 4th World Forum on Internet of things (WF-IoT)*. IEEE. 2018, pp. 729–735.

[33] Davis Blalock, Samuel Madden, and John Guttag. "Sprintz: Time series compression for the internet of things". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2.3 (2018), pp. 1–23.

[34] Ali Kadhum Idrees et al. "Distributed Data Aggregation based Modified K-means technique for energy conservation in periodic wireless sensor networks". In: *2018 IEEE Middle East and North Africa Communications Conference (MENACOMM)*. IEEE. 2018, pp. 1–6.

[35] Shijing Li et al. "EF-Dedup: Enabling Collaborative Data Deduplication at the Network Edge". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2019, pp. 986–996.

[36] Elena Fasolo et al. "In-network aggregation techniques for wireless sensor networks: a survey". In: *IEEE Wireless Communications* 14.2 (2007), pp. 70–87.

[37] Abdallah Makhoul and Hassan Harb. "Data reduction in sensor networks: Performance evaluation in a real environment". In: *IEEE Embedded Systems Letters* 9.4 (2017), pp. 101–104.

[38] Tao Du et al. "A high efficient and real time data aggregation scheme for wsns". In: *International journal of distributed sensor networks* 11.6 (2015), p. 261381.

[39] Song Lin et al. "Region sampling: Continuous adaptive sampling on sensor networks". In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE. 2008, pp. 794–803.

[40] Zengfeng Huang et al. "Sampling based algorithms for quantile computation in sensor networks". In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 2011, pp. 745–756.

[41] Hassan Harb et al. "A distance-based data aggregation technique for periodic sensor networks". In: *ACM Transactions on Sensor Networks (TOSN)* 13.4 (2017), pp. 1–40.

[42] Joseph Azar et al. "On the performance of resource-aware compression techniques for vital signs data in wireless body sensor networks". In: *2018 IEEE Middle East and North Africa Communications Conference (MENACOMM)*. IEEE. 2018, pp. 1–6.

[43] Minh Tuan Nguyen, Keith A Teague, and Nazanin Rahnavard. "CCS: Energy-efficient data collection in clustered wireless sensor networks utilizing block-wise compressive sensing". In: *Computer Networks* 106 (2016), pp. 171–185.

[44] Xiao-Yang Liu et al. "CDC: Compressive data collection for wireless sensor networks". In: *IEEE Transactions on Parallel and Distributed Systems* 26.8 (2014), pp. 2188–2197.

[45] Dnyaneshwar Mantri, Neeli Rashmi Prasad, and Ramjee Prasad. "BHCDA: Bandwidth efficient heterogeneity aware cluster based data aggregation for Wireless Sensor Network". In: *2013 international conference on advances in computing, communications and informatics (ICACCI)*. IEEE. 2013, pp. 1064–1069.

[46] Wong Siaw Ling, Ooi Boon Yaik, and Liew Soung Yue. "A novel data reduction technique with fault-tolerance for internet-of-things". In: *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*. 2017, pp. 1–7.

[47] Zaid Yemeni et al. "Reliable spatial and temporal data redundancy reduction approach for wsn". In: *Computer Networks* 185 (2021), p. 107701.

[48] Waleed M Ismael, Mingsheng Gao, and Zaid Yemeni. "ESRRA-IoT: Edge-based spatial redundancy reduction approach for Internet of Things". In: *Internet of Things* 14 (2021), p. 100388.

[49] MK Alam et al. "Data Clustering Technique for In-Network Data Reduction in Wireless Sensor Network". In: *2019 IEEE Student Conference on Research and Development (SCOReD)*. IEEE. 2019, pp. 317–322.

[50] Sakil Chowdhury and Abderrahim Benslimane. "Relocating redundant sensors in randomly deployed wireless sensor networks". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2018, pp. 1–6.

[51] Koustuv Dasgupta, Konstantinos Kalpakis, and Parag Namjoshi. "An efficient clustering-based heuristic for data gathering and aggregation in sensor networks". In: *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.* Vol. 3. IEEE. 2003, pp. 1948–1953.

[52] MK Alam et al. "Error-Aware Data Clustering for In-Network Data Reduction in Wireless Sensor Networks". In: *Sensors* 20.4 (2020), p. 1011.

[53] Prakashgoud Patil and Umakant Kulkarni. "SVM based data redundancy elimination for data aggregation in wireless sensor networks". In: *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE. 2013, pp. 1309–1316.

[54] Ata Ullah et al. "Secure Healthcare Data Aggregation and Deduplication Scheme for FoG-Orineted IoT". In: *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE. 2019, pp. 314–319.

[55] Ata Ullah et al. "FoG assisted secure De-duplicated data dissemination in smart healthcare IoT". In: *2018 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE. 2018, pp. 166–171.

[56] Farah Karim et al. "Large-scale storage and query processing for semantic sensor data". In: *Proceedings of the 7th international conference on web intelligence, mining and semantics*. 2017, pp. 1–12.

[57] Yuan Lingyun et al. "RFID data fusion algorithm based on spatio-temporal semantics in internet of things". In: *2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*. IEEE. 2017, pp. 179–184.

[58] Hafizur Rahman, Nurzaman Ahmed, and Iftekhar Hussain. "Comparison of data aggregation techniques in Internet of Things (IoT)". In: *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE. 2016, pp. 1296–1300.

[59] Ioannis K Vlachos and George D Sergiadis. "Intuitionistic fuzzy information–applications to pattern recognition". In: *Pattern Recognition Letters* 28.2 (2007), pp. 197–206.

[60] Lotfi A Zadeh. "Making computers think like people [fuzzy set theory]". In: *IEEE spectrum* 21.8 (1984), pp. 26–32.

[61] Timothy J Ross. *Fuzzy logic with engineering applications*. John Wiley & Sons, 2005.

[62] Bernadette Bouchon-Meunier et al. "Compositional rule of inference as an analogical scheme". In: *Fuzzy Sets and Systems* 138.1 (2003), pp. 53–65.

[63] George Salloum and Joe Tekli. "Automated and personalized nutrition health assessment, recommendation, and progress evaluation using fuzzy reasoning". In: *International Journal of Human-Computer Studies* 151 (2021), p. 102610.

[64] Pablo Cingolani and Jesus Alcala-Fdez. "jFuzzyLogic: a robust and flexible Fuzzy-Logic inference system language implementation". In: *2012 IEEE International Conference on Fuzzy Systems*. IEEE. 2012, pp. 1–8.

[65] Pablo Cingolani and Jesús Alcalá-Fdez. "jFuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming". In: *International Journal of Computational Intelligence Systems* 6.sup1 (2013), pp. 61–75.

[66] Bodik P. et al. *Intel Lab Data*. http://db.csail.mit.edu/labdata/labdata.html. Accessed: 2021-09-24.

[67] Fekade Getahun Taddesse et al. "Relating RSS news/items". In: *ICWE'9 Proceedings of the 9th International Conference on Web Engineering*. 2009, pp. 442–452.

[68] Marc Ehrig and York Sure. "Ontology mapping–an integrated approach". In: *European semantic web symposium*. Springer. 2004, pp. 76–91.

[69] Ralph Abboud and Joe Tekli. "Integration of nonparametric fuzzy classification with an evolutionary-developmental framework to perform music sentiment-based analysis and composition". In: *Soft Computing* 24.13 (2020), pp. 9875–9925.

[70] Danielle Azar, Karl Fayad, and Charbel Daoud. "A combined ant colony optimization and simulated annealing algorithm to assess stability and fault-proneness of classes based on internal software quality attributes". In: *International Journal of Artificial Intelligence* 14.2 (2016), pp. 137–156.

[71] Hussein Jebbaoui et al. "Semantics-based approach for detecting flaws, conflicts and redundancies in XACML policies". In: *Computers & Electrical Engineering* 44 (2015), pp. 91–103.

[72] Ramzi A Haraty and Mohamed El Sai. "Information warfare: a lightweight matrix-based approach for database recovery". In: *Knowledge and Information Systems* 50.1 (2017), pp. 287–313.

[73] Ramzi A Haraty, Mirna Zbib, and Mehedi Masud. "Data damage assessment and recovery algorithm from malicious attacks in healthcare data sharing systems". In: *Peer-to-Peer Networking and Applications* 9.5 (2016), pp. 812–823.