Using Fuzzy Reasoning to Improve Redundancy Elimination for Data Deduplication in Connected Environments

Sylvana Yakhni
E.C.E. Department,
Lebanese American Univ.
36 Byblos, Lebanon
silvana.yakhni@lau.edu

Joe Tekli
E.C.E. Department,
Lebanese American Univ.
36 Byblos, Lebanon
joe.tekli@lau.edu.lb

Elio Mansour
Univ. Pau & Pays Adour,
E2S UPPA, LIUPPA
Anglet, 64600, France
elio.mansour@univ-pau.fr

Richard Chbeir
Univ. Pau & Pays Adour,
E2S UPPA, LIUPPA
Anglet, 64600, France
richard.chbeir@univ-pau.fr

Abstract—**The Internet of things (IoT) is ushering in the era of connected environments where the number and diversity of data sources (devices and sensors) are inevitably increasing the size of the data that need to be stored locally (at the edge device level) and transmitted to base storages (at the sink level) of the network. This huge amount of data highlights several challenges including network bandwidth, consumption of network energy, cloud storage, and I/O throughput. These call for data pre-processing and filtering solutions to reduce the amount of data being handled and transmitted over the network. In this study, we investigate data deduplication as a prominent pre-processing method that can be used and adapted to address such challenges. Data deduplication techniques have been traditionally developed for data storage and data warehousing applications, and aim at identifying and eliminating redundant data items. Few recent approaches have been designed for connected environments, yet they share various limitations, including: i) detecting duplicates at one level only of the network (either edge or sink exclusively), ii) overlooking the context and dynamicity of the network (disregarding device mobility, and overlooking boundary separations and sensor coverage areas), iii) relying on crisp thresholds and providing minimum-to-no expert control over the deduplication process (disregarding the domain expert's needs in defining redundancy). In this study, 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 domain knowledge and expert preferences regarding data duplication boundaries. It then applies pattern codes and fuzzy reasoning to detect duplicates at both the edge level and the sink level of the network. This reduces the time required to hard-code the deduplication process, while adapting to the domain expert's needs for different data sources and applications. Moreover, FREDD is adapted for multiple scenarios, considering both static and mobile devices, with different configurations of hard-separated and soft-separated zones, and different sensor coverage areas in the connected environment. Experiments on a real-world dataset highlight FREDD's potential and improvement compared with existing solutions.**

Keywords—**Connected Environments, Fuzzy Reasoning, Data Redundancy, Data Deduplication, Internet of Things (IoT), Cyber-Physical Systems, Wireless Sensor Networks.**

1. Introduction

Recent advances in data management and sensing technologies have allowed physical infrastructures (e.g., buildings, homes, and cities) to become more connected (Li D. *et al.* 2019, Kaur R. *et al.* 2018). In fact, the Internet of Things (IoT) is ushering-in the era of connected environments, i.e., networks of physical objects that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems (e.g., smart hospitals, smart buildings, and smart cities) (Nižetić S. et al. 2020, Lytras M. *et al.* 2015). These connected environments produce huge amounts of sensed data that can be exploited for various high-level applications. According to a recent survey conducted by VoucherCloud in (VoucherCloud 2018), 2.5+ quintillions of sensory data are currently generated every day, and over 123×10^9 IoT devices are expected to be connected within the next 10 years (IoT Analytics 2021). This huge amount of data highlights several challenges including network bandwidth, consumption of network energy, cloud storage, and I/O throughput. These call for data pre-processing and filtering techniques to reduce the amount of data being handled and transmitted over the network. In this study, we investigate data deduplication as a prominent pre-processing method that can address such challenges. Data Deduplication techniques have been traditionally developed for data storage and data warehousing applications, and aim at identifying and eliminating redundant data items, where only one unique copy of the data is stored (Ismael W. *et al.* 2019). Similarly, data deduplication in connected environments aims at eliminating redundant measurements produced by sensing devices. For instance, there is no need to store and transmit similar temperature measurements produced by a sensor if they are almost identical within a given timespan. Likewise, there is no need to process measurements produced by one or multiple sensors located in a certain geographic area, as long as the area temperature has not significantly changed. A similar behavior is expected from mobile sensors moving within a given area, or crossing from one area to another, given certain area and network constraints (cf. motivation scenario in Section 2.1). Such measurements would be considered redundant and need to be eliminated, where only relevant changes are processed by the system. In this context, few recent approaches have been designed for handling data in connected environments, e.g., (Mansour E. *et al.* 2020, Ismael W. *et al.* 2019, Li S. et al. 2019, Patil P. and Kulkarni U. 2013, Qutub B. *et al.* 2012), yet they share various limitations. First, they work on one single level: i) at the edge device (sensor) level only, e.g., (Mansour E. *et al.* 2020, Qutub B. *et al.* 2012),

disregarding redundancies at the sink level (base station) or at the network core, ii) at the sink level only, e.g., (Ullah A. et al. 2019, Patil P. and Kulkarni U. 2013), resulting in increased bandwidth and network energy consumption when transmitting the raw data from the edge device to the sink device or to the network core. Second, existing solutions mostly disregard the dynamicity and constraints of the network, considering static devices only (e.g., stationary surveillance cameras, pollution sensors), e.g., (Ismael W. *et al*. 2019, Li S. et al. 2019, Patil P. and Kulkarni U. 2013), overlooking mobile devices (e.g., phones, tablets), and overlooking boundary separations (e.g., hard-separated and soft-separated zones) and sensor coverage areas (which may differ from a sensor to another). Third, most existing solutions rely on crisp evaluation thresholds and provide the domain expert with minimum-to-no control over the deduplication process (i.e., which data need to be duplicated and which data should be kept intact) hence overlooking the expert's requirements and application needs in defining redundancy, e.g., (Mansour E. *et al.* 2020, Patil P. and Kulkarni U. 2013).

In this study, 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 domain knowledge and expert preferences regarding data duplication boundaries. It then applies pattern codes and fuzzy reasoning to detect duplicates on the general network infrastructure including both the edge level and the sink level of the network. This reduces the time required to hard-code the deduplication process, while adapting to the domain expert needs and application requirements. Moreover, FREDD is adapted for multiple scenarios, considering both static and mobile devices, with different configurations of hard-separated and soft-separated zones, and different sensor coverage areas in the connected environment. Experiments on a real-world dataset highlight FREDD's potential and improvement compared with existing solutions.

The remainder of this paper is organized as follows. Section 2 presents the background and motivations of our study. Section 3 briefly reviews the literature and related works. Section 4 describes the FREDD framework and its modules. Section 5 describes different deduplication use-cases and the approach to handle each case. Section 5 describes our experimental evaluation and results. Finally, Section 7 concludes with future directions.

2. Background and Motivations

2.1. Motivation Scenario

Consider the following example that illustrates a smart hospital, describing the motivations, needs, and challenges behind this work. Figure 1 depicts two wards of the hospital, each hosting a set of static sensing devices that provide *humidity* and *temperature* measurements (among others) from the environment. Similar measurements are also provided by mobile devices such as smart phones and tablets operated by the medical staff. Devices have built-in memories to buffer chunks of sensory data before transmission to the network's sink nodes. Every ward of the hospital has a specific set of locations/rooms separated by either hard physical entities (e.g., walls, windows) or soft entities (e.g., open cubicles, curtains). The type of separation (i.e., hard or soft) is decided based on health and safety requirements (e.g., walls for the Intensive Care Unit -ICU, cubicles for staff offices). Every ward has a static sink node that aggregates all the data from its connected edge devices into the sink node storage memory[1].

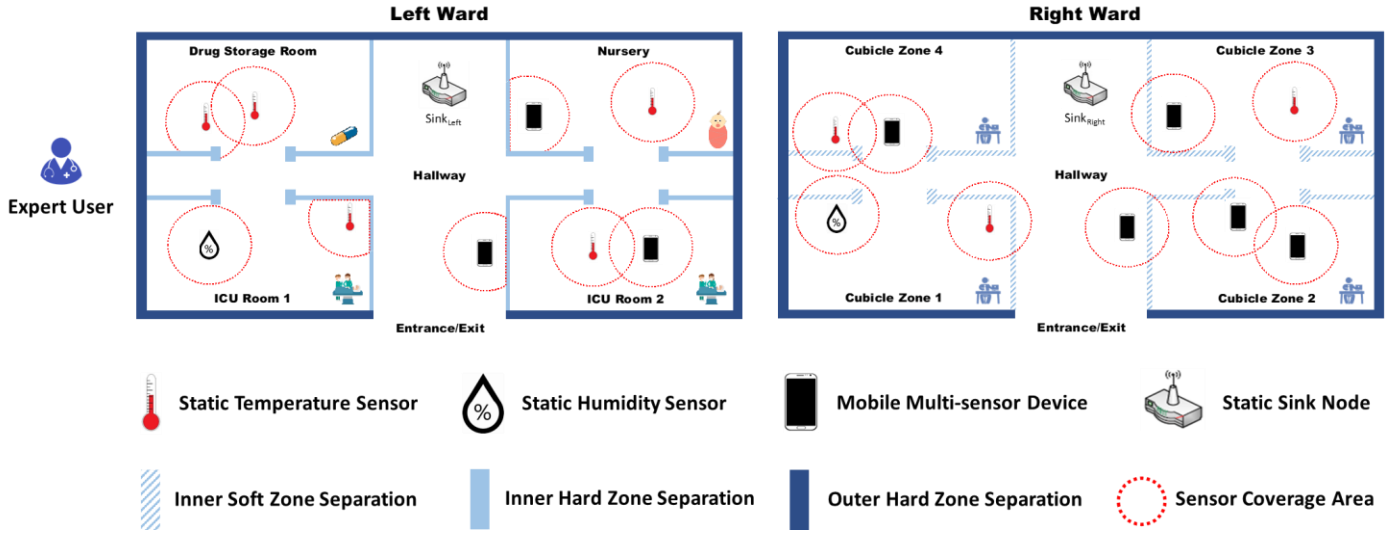


Figure 1. Motivating scenario example of a smart hospital connected environment.

[1] In this scenario, we disregard inter-sink node connectivity (i.e., inter-sink node collaboration) and their connectivity with the network's base station. The latter configurations bring forth additional challenges including sink node mobility, sink node coverage area overlapping, and composite redundancies by data fusion from multiple edged and sink nodes, which we report to a future dedicated study.

Technicians need to monitor the hospital environment to regulate air quality in specific areas (e.g., ICU, nursery) and to ensure the prevention of overheating or underheating in certain areas (e.g., drug storage room). To do so, we highlight the following needs: *need 1* - retrieving non-redundant and concise *humidity/temperature* data from individual locations (i.e., directly from the edge devices) and from entire zones (i.e., from the wards' sink nodes); *need 2* - retrieving non-redundant and concise *humidity/temperature* while considering device mobility (e.g., data from mobile devices like medical staff tablets, smartphones), spatial constraints of the hospital (e.g., locations and zone separations) and their impact on sensing/sensor coverage areas (e.g., sensor coverage areas are blocked by walls but not by open cubicles); *need 3* - defining health-related constraints for specific areas in the hospital (e.g., normal value range and significant variations of *humidity* levels in the nursery, normal value range and significant variations of *temperature* measurements in the drug storage room and staff offices) and calibrating data storage and retrieval based on the aforementioned constraints.

In this setup, the sensing devices produce and exchange huge amounts of data that often contain redundancies (i.e., duplicate values that are not necessary for processing and storage). For instance, if *humidity* is stable in a specific room, multiple unnecessary duplicate values are sent to the sink. This can exhaust network and device resources, as it requires more resources for data processing, storage, and retrieval. While many duplicate measurements would qualify as useless data redundancies, yet this is not the case for all duplicates, especially when the measurements are made by the same mobile sensor moving between different locations, or when the measurements are made by different sensors at different locations in the hospital (identical measurements made at the same time in two different rooms do not qualify as redundancies at the room level, yet they might qualify as redundancies at the floor or ward level including both rooms; also, mobile devices can be located in the same ward yet separated by an opaque wall, such that their measurements are unrelated and would not qualify as redundancies). In this context, data deduplication can be utilized to answer the aforementioned needs: allowing to properly detect and eliminate uselessly redundant measurements, taking into account the different device and environment properties and constraints, in order to increase storage capacities and improve performance without affecting data processing and retrieval accuracy. More specifically, we consider the following challenges mapping to the aforementioned needs: *challenge 1* - how to accurately detect data redundancies and remove them at the device and sink levels; *challenge 2* - how to consider the physical constraints of the environment when deduplicating data (e.g., device mobility, spatial constraints, soft and hard zone separations, and sensor coverage areas); *challenge 3* - how to configure the deduplication process in order to inject domain insights and knowledge (e.g., assigning different membership scores to certain measurement variations and defining variation boundaries, based on expert preferences) for a more adapted and accurate redundancy detection process.

In this work, we tackle data deduplication in connected environments and design our framework solution based the fuzzy logic paradigm to address the aforementioned challenges.

2.2. Preliminaries on Data Deduplication

Devices/sensors in connected environments are often densely deployed to monitor the environment and report observations. These devices generate huge amounts of redundant data by sensing the environment. Sensed data is often spatially and temporally tagged. This allows an elaborate evaluation of inconsistencies that might result from temporal and spatial data redundancies. A temporal redundancy is caused by a single stationary sensor producing redundant observations at different time stamps (e.g., a temperature sensor producing identical or similar measurements in a certain continuous period of time). A spatial redundancy is caused by a group of stationary sensors deployed in close vicinity of each other and producing redundant observations (e.g., multiple temperature sensors located in the same room and producing identical or similar measurements). A spatial-temporal redundancy is produced by mobile sensors/devices which are moving around the network and generating redundant observations at different time stamps and locations simultaneously. In this context, most existing works in the literature perform deduplication on identical data (e.g., measurements 18.2°C at time *t* and 18.2°C at time *t+1* are considered to be redundant and are thus duplicated by eliminating one measurement and keeping the other), yet they do not address data similarities (e.g., 18.2°C at *t* and 18.3°C at *t+1* are considered to be different measurements, and are consequently processed, stored, and transmitted across the network, cf. Section 3). In this study, we highlight the need to consider both identical and similar data items in performing deduplication, since data fluctuations are common in connected environments due to the real-world conditions and constraints of sensing devices (e.g., 18.2°C at *t* and 18.3°C at *t+1* might represent unintentional fluctuations in temperature measurements due to sensing device accuracy or domain constraints, and thus can to be considered redundant at the data level and processed for deduplication). As a result, we adopt fuzzy reasoning to detect duplicates among similar data measurements, considering device specs, domain knowledge, and expert preferences in defining the data deduplication boundaries. We provide a sneak peek on fuzzy reasoning in the following subsection.

2.3. Preliminaries on Fuzzy Logic

Fuzzy logic is a multivalued 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 usually incorporates a condition-action rule-based *IF X AND Y THEN Z* approach rather than attempting to model a system mathematically (Ross T. J. 2016). 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 (Vlachos I. K. and Sergiadis G. D. 2007). For example, rather than dealing

with data values in terms of humidity = 95 µg/m^3 and temperature = 18.2 °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 (Ross T. J. 2016, Zadeh L. A. 1984): 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/m^3 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 domain 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.

In this study, 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. We review next the related works, before describing and evaluating our proposal.

3. Related Works

Data deduplication techniques have been widely researched in data storage and data warehousing systems, and have been recently investigated in the context of IoT and connected environments.

3.1. 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[2] (e.g., VMWare ESX, Linux KSM), and SSDs[3] (e.g., Cache Acceleration Software CAS by CAFTL) (Paulo J. and Pereira J. 2014). 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 (Malhotra J. and Bakal J. 2015). 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 (Bhalerao A. and Pawar A. 2017, Malhotra J. and Bakal J. 2015). Data deduplication is also a necessary step in data cleaning, also referred to as *data scrubbing* in data warehousing (Christen P. 2012). 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 subqueries, temporary tables, derived tables, Common Table Expressions (CTEs), and dynamic SQL (Attigeri G. *et al.* 2010). 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 (Christen P. 2012). Some of the used clustering techniques include Sorted Neighborhood, Q-gram based clustering, and Canopy clustering (Christen P. 2012).

Discussion: Most deduplication techniques for data storage and data warehousing assume textual data duplicates only and disregard numerical values, e.g., (Bhalerao A. and Pawar A. 2017, Malhotra J. and Bakal J. 2015, Christen P. 2012). The few methods which address numerical data, e.g., (Attigeri G. *et al.* 2010) 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 environment, where most data collected from sensors are scalar.

3.2. Deduplication in Connected Environments

We categorize deduplication approaches in connected environments following the challenges described in our motivation scenario (cf. Section 2.1): *challenge 1* - considering redundancies at the edge or at the sink: stating if the approach handles data redundancy at the source (device level) and if it prevents redundancies from reaching the core; *challenge 2* - considering the dynamicity and physical constraints of the environment, including: device mobility, zone separations, and sensor coverage areas; and *challenge 3* - considering expert-centric data deduplication: specifying if the approach considers the domain experts' needs when removing redundancies.

In (Patil P. and Kulkarni U. 2013), 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

[2] Random Access Memory
[3] Solid State Drive

in a central node, which provides a redundancy-free data repository that can be mined using dedicated data processing techniques (cf. *challenge 1*). However, redundancies are not handled at the edge level, and data exchange between devices at the edge remains costly due to unnecessary communications. In (Ullah A. et al. 2019), the authors provide a data deduplication technique in healthcare-based IoT, and introduce a Controlled Window-size based Chunking Algorithm (CWCA) to identify cut-points in sensor data distributions. Yet similarly to (Patil P. and Kulkarni U. 2013), the solution in (Ullah A. et al. 2019) only performs data deduplication at sink nodes and does consider redundancies at edge devices (cf. *challenge 1*). Moreover, the solutions in (Ullah A. et al. 2019, Patil P. and Kulkarni U. 2013) do not consider spatial-temporal redundancies generated by mobile devices (cf. *challenge 2*). In (Ismael W. *et al.* 2019), the authors present a data reduction scheme using data filtering and fusion. They handle redundancies at the edge device level before forwarding non-redundant data to the sink level. Redundancy detection is based on data value deviations only, and does not consider redundant data from mobile devices (cf. *challenge 2*). In (Chowdhury S. and Benslimane A. 2018), the authors focus on the spatial distribution of sensors in the environment, and how it can be managed to prevent redundancies. The authors build a graph of nodes and events in order to detect "redundant" nodes: i.e., nodes producing identical events. Consequently, redundant nodes are either relocated or put into sleep mode using a circle packing technique to enhance coverage, while minimizing energy usage during relocation. This work only handles redundancy from a sensor deployment perspective (i.e., avoiding deploying sensors that provide the same type of data in the same area). The approach focuses on detecting redundant sensor nodes rather than the data itself, and does not consider sensor mobility (cf. *challenge 2*). The authors in (Li S. et al. 2019) propose a data redundancy elimination 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 (cf. *challenge 2*).

In a continuous sensing setup, triggering mechanisms are available to restrict the number of transmissions between the sensor node and the monitoring node without degrading the tracking of the sensed measurements, e.g., (Liansheng T. and Wu M. 2015, Santini S. and Romer K. 2006). These mechanisms can also be used for filtering redundant spatial-temporal data, in order to trigger transmissions from sensor node (edge) to monitoring node (sink) only when there are changes in the sensed measurements. These approaches fall into the category of edge-based data deduplication solutions, and rely on a simple crisp deviation threshold δ_v. Therefore, if consecutive measurements at time *t* and *t+1* are higher than the threshold, $|v_t - v_{t+1}| \geq \delta_v$, the edge sends the data to the sink. If not, duplicates are eliminated and the oldest value is usually flushed. In this context, relying on crisp thresholds can lead to deduplication accuracy drops, where even the slightest variations in the sensed measurements are processed similarly to extremely large variations (e.g., given a temperature variation threshold δ_v = 1°C, variations of 1.5 °C and 20.5 °C are processed exactly the same). Similarly, variations which are slightly below the variation threshold will be completely ignored (e.g., given a temperature variation threshold δ_v=1°C, a variation of 0.99 °C goes unprocessed). Hence, relying on crisp thresholds restricts domain insights and expert knowledge (cf. *challenge 3*), and might lead to i) missing certain relevant redundancies or ii) removing certain data values that might not be redundant.

More recently, the authors in (Mansour E. *et al.* 2020) proposed a Data Redundancy Management Framework (DRMF) that handles data redundancies at the edge device level (cf. *challenge 1*) considering both static and mobile devices (cf. *challenge 2*). It clusters the data based on an expert/system defined crisp deviation threshold (cf. *challenge 3)*, while also keeping track of the temporal and spatial-temporal spread (or coverage) of each cluster (i.e., sets of redundant data). The algorithm sorts all data tokens and checks if the current token belongs to the current cluster by comparing it with the cluster's centroid, considering a expert-defined deviation threshold δv. Otherwise, a new cluster is created with the current data item added as its centroid. In another relevant study in (Khriji S. *et al.* 2018), the authors introduce the Redundancy Elimination Data Aggregation (REDA) algorithm to perform deduplication at individual edge nodes and also among nodes in the same cluster (cf. *challenge 1*). Assuming that data is presented as a set of scalar values (e.g., temperature, humidity), the range of numbers is divided into crisp intervals that depend on the domain expert requirements (cf. *challenge 3*), then a lookup table is generated for each cluster containing the ranges of each interval and their associated pattern codes (e.g., 10-15°C are associated with pattern code 1, 16-20° are associated with pattern code 2). For the first iteration (t=0), each sensor (edge) computes and sends its own pattern code to the cluster head (sink) where only a unique code is saved. For the following iterations (t>0), each senor computes its new pattern code and compares it with the last one. The new value is sent from the sensor (device) to the cluster head (sink) only if its pattern code is different from that of the cluster head. Note that both (Mansour E. *et al.* 2020, Khriji S. *et al.* 2018) rely on crisp deviation thresholds, and share the same limitations of crisp processing mentioned above.

3.3. Discussion

To wrap up, we highlight the main issues facing data deduplication methods in connected environments, in light of the main challenges considered in our study. Considering *challenge 1*, most approaches focus on handling redundancies at the edge level only (Mansour E. *et al.* 2020, Li S. et al. 2019) or at the sink (core) level only (Murtadha H. and Sami S. 2016, Mortadha H. and Jihad A. 2015, Attigeri G. *et al.* 2010, Shahri H. and Barforush A. 2004). Very few methods address both edge and sink data redundancies (Ismael W. *et al.* 2019, Papageorgiou A. *et al.* 2015, Patil P. and Kulkarni U. 2013), and they do so in specific scenarios. For instance, in (Khriji S. *et al.* 2018), the authors consider clusters of geographically neighboring sensors that elect cluster heads (as sink nodes) where deduplication takes place. However, this design choice is not necessarily applicable when sink nodes are not bound by geographic proximity (e.g., temperature sensors in different building floors can be connected to the same

sink, however this does not mean that data from all these sensors can be considered for deduplication). Considering *challenge 2*, most existing solutions consider the case of static (stationary) devices only and disregard mobile devices (Ismael W. *et al.* 2019, Li S. et al. 2019, Papageorgiou A. *et al.* 2015, Patil P. and Kulkarni U. 2013). Only one approach provided in (Mansour E. *et al.* 2020) handles device mobility, considering data measurement location (in the form of spatial-temporal redundancies). Yet, the authors in (Mansour E. *et al.* 2020) do not address zone separations (hard/soft) and device coverage areas (coverage radius). Considering *challenge 3*, most existing works focus solely on the network or device properties in designing their deduplication solutions, and perform crisp deduplication processing using crisp evaluation thresholds, thus providing minimum-to-no expert intervention or adaptability when eliminating redundancies.

4. FREDD Framework

To address the aforementioned challenges, we introduce FREDD: a new framework for Fuzzy Redundancy Elimination for Data Deduplication in a connected environment. FREDD detects data duplicates at both edge and sink levels (*criterion 1*), considers data redundancies from static and mobile devices with varying boundaries and coverage areas (*criterion 2*), and combines simple natural language rules with a fuzzy inference mechanism designed to adapt the deduplication process following the expert's needs (*criterion 3*). FREDD's core architecture for edge-level deduplication is depicted in Fig. 1. It consists of six main modules: i) **sensor data representation** which defines the spatial and temporal representations of data measurements/items, ii) **measurement separation** which separates the input data into measurement-based data collections, iii) **pattern code generation** which associates data items with pattern codes based on expert-defined lookup tables, iv) **duplicate candidate filtering** which determines whether data items are candidates for fuzzy duplication, v) **fuzzy redundancy detection** which identifies duplicate data items using fuzzy reasoning based on expert-defined condition-action rules, and vi) **redundancy removal** which eliminates redundant data items and produces deduplicated data. We describe each of the latter modules for edge-level deduplication in the following sub-sections. FREDD's handling of sink-level deduplication use-cases is subsequently described in Section 5.

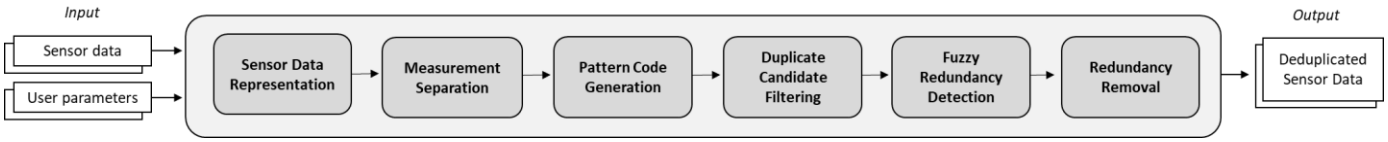


Figure 2. Simplified diagram describing FREDD's core architecture

4.1. 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. Here, we adopt a set of formal definitions from (Mansour E. *et al.* 2020) that allow us to describe data items following both temporal and spatial dimensions (cf. *criterion 2*).

Definition 1 - Data Items: A data item d is defined as a 5-tuple:

$$d = \langle m\ ;\ v\ ;\ t\ ;\ l\ ;\ s \rangle \quad \textbf{(1)}$$

where m is the data measurement, v is the data value, t is the creation temporal stamp of d (cf. Definition 2), l is the creation location stamp of d (cf. Definition 3), and s is the data source that sensed/created d ●

Definition 2 - Temporal Stamp: A temporal stamp t is defined as a single discrete temporal value represented as a 2-tuple:

$$t = \langle format\ ;\ value \rangle \quad \textbf{(2)}$$

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

Definition 3 – Location Stamp: A location stamp l is defined as a discrete and instantaneous location value represented as a 2-tuple:

$$l = \langle format\ ;\ value \rangle \quad \textbf{(3)}$$

where *format* is the location referential format of the location value (e.g., default GPS, or Cartesian, Spherical, Cylindrical), and $value = \langle x\ ;\ y\ ;\ z \rangle$ 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) ●

Table 1 shows the representation of sample sensory data after being sensed/produced by an edge device (source) S_1 embedding two sensors producing *humidity* and *temperature* measurements respectively.

Table 1. Sample sensory data items

Measurement *m*	**Value *v***	**Time stamp *t***		**Location stamp *l***				**Source *s***
		format	*value*	*format*	*value*			
					x	y	z	
Humidity	92 µg/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	S1
Humidity	94 µg/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	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	Cartesian	8	6	3	S1
Humidity	103 µg/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	S1
Humidity	104 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	Cartesian	7	2	7	S1

4.2. Measurement Separation

Since the device can embed various sensors, its internal memory might store different measurements (i.e., features such as *humidity* and *temperature* in Table 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 measurements. To illustrate the measurement filtering process, the data shown in Table 1 produces two distinct data collections: the first for *humidity* data (first four tuples - cf. Table 2), and the second for *temperature* data (containing the last tuples). Consequently, the measurement data collections are processed separately for data deduplication. Note that the domain expert decides about the selection of measurements to be processed for deduplication.

Table 2. Data collections produced following the measurement separation of the data from Table 1

a. *Humidity* data collection

Measurement *m*	**Value *v***	**Time stamp *t***		**Location stamp *l***				**Source *s***
		format	*value*	*format*	*value*			
					x	y	z	
Humidity	92 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	Cartesian	1	2	3	S1
Humidity	94 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	Cartesian	4	2	7	S1
Humidity	103 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	5	2	7	S1
Humidity	104 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:06:00	Cartesian	7	2	7	S1

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Time stamp *t***		**Location stamp *l***				**Source *s***
		format	*value*	*format*	*value*			
					x	y	z	
Temperature	16 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	2	5	3	S1
Temperature	19.5 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	Cartesian	5	6	3	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	Cartesian	8	6	3	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	8	6	3	S1

4.3. Pattern Code Generation

4.3.1. 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 kings of lookup tables allowing: i) *disjoint* data ranges, and ii) *intersecting* data ranges.

Table 3. Sample *disjoint* value lookup tables for *humidity* and *temperature* measurements, considering ranges 90-110 µg/m [4] and 15-28 °C respectively

a. Disjoint *humidity* data ranges

Interval values	[90, 96] µg/m^3	]96, 104] µg/m^3	]104, 110] µg/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. Sample *intersecting* value lookup tables for *humidity* and *temperature* measurements, considering ranges 90-110 µg/m^3 and 15-28 °C respectively

a. Intersecting *humidity* data ranges

Interval values	[90, 98] µg/m^3	[94, 106] µg/m^3	[102, 110] µg/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 3) 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 µg/m^3 or 104.7 µg/m^3 following Table 3). *Intersecting* data ranges (cf. Table 4) 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 µg/m^3 belong to both H2 and H3 patter codes following Table 4).

In this study, we consider *intersecting* ranges to allow more efficient processing (duplicate candidate filtering, cf. Section 4.4) and more accurate data deduplication (fuzzy redundancy detection, cf. Section 4.5)

4.3.2. 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 and sink 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 5 shows a sample zone lookup table describing a sample zoning in our smart hospital example shown in Figure 3, 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.

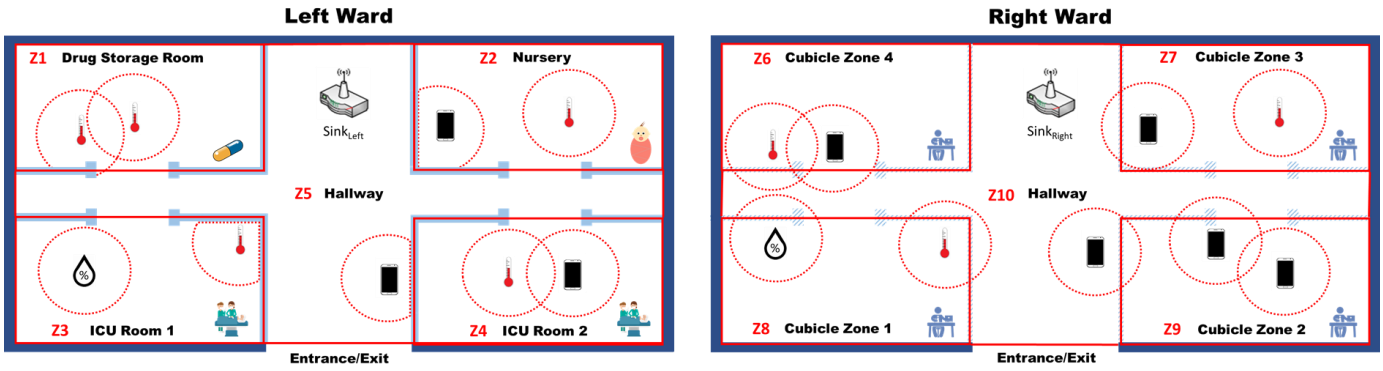


Figure 3. Zone divisions in our smart connected hospital motivation example (cf. Section 2.1)

Table 5. Zone lookup table for Sink$_{left}$ in the smart connected hospital example from Figure 3

Interval values	$\{p_1, \ldots, p_k\}$	$\{p_{k+1}, \ldots, p_m\}$	$\{p_{m+1}, \ldots, p_n\}$	$\{p_{n+1}, \ldots, p_o\}$	$\{p_{o+1}, \ldots, p_p\}$
Label	Drug Storage Room	Nursery	ICU Room1	ICU Room2	Hallway
Pattern code	Z1	Z2	Z3	Z4	Z5

[4] Microgram Per Cubic Meter

Table 6. Zone lookup table for $Sink_{Right}$ in the smart connected hospital example from Figure 3

Interval values	$\{p_{p+1}, \ldots, p_q\}$	$\{p_{q+1}, \ldots, p_r\}$	$\{p_{r+1}, \ldots, p_s\}$	$\{p_{s+1}, \ldots, p_t\}$	$\{p_{t+1}, \ldots, p_u\}$
Label	Cubicle Zone4	Cubicle Zone3	Cubicle Zone 1	Cubicle Zone2	Hallway
Pattern code	Z6	Z7	Z8	Z9	Z10

4.3.3. 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 Table 7 (cf. Algorithm 1 in Figure 4). The value-zone codes are used for fast duplicate candidate filtering as described in the following section.

Table 7. Value, zone, and combined pattern codes for sample data from Table 1.

a. *Humidity* data collection

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Location stamp *l***				**Zone Pattern Code**	**Combined Pattern Code**	**Source *s***
			format	*value*	*format*	*value*					
						x	y	z			
Humidity	92 µg/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 µg/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 µg/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 µg/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

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Location stamp *l***				**Zone Pattern Code**	**Combined Pattern Code**	**Source *s***
			format	*value*	*format*	*value*					
						x	y	z			
Temperature	16 °C	{T1}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	2	5	3	Z2	{T1_Z2}	S1
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}	S1
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	Cartesian	8	6	3	Z4	{T2_Z4}	S1
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	8	6	3	Z4	{T2_Z4}	S1

4.4. 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 Figure 4. 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 value-zone pattern code, then they are considered duplicates (cf. Figure 4, lines 4-5), ii) if the data items share one or more value-zone pattern codes, they are considered as candidates for deduplication (cf. Fig. , lines 6-7), and iii) if the data items do not share any value-zone pattern code, they are considered as non-duplicates (cf. Figure 4, lines 8-9),

Table 8 shows the output of the filtering algorithm applied on the input data from Table 7, where 6 data items are identified as either duplicates/non-duplicates, such that 2 of the original 8 items need to be further considered for fuzzy deduplication. 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 7).

Algorithm 1 – Duplicate Candidate Filtering

Input: DataItem1, DataItem2
Output: DeduplicationStatus

```
Begin
 1    pattern1 ← pattern code for DataItem1
 2    pattern1 ← pattern code for DataItem2
 3    interLen ← length of intersection between DataItem1 and DataItem2
 4    if (pattern1 = pattern2) and (interLen =1) then
 5         DeduplicationStatus ← Duplicates
 6    else if interLen > 1 then
 7         DeduplicationStatus ← Candidates
 8    else
 9         DeduplicationStatus ←  NotDuplicates
End
```

Figure 4. Pseudocode of our *duplicate candidate filtering* algorithm

Table 8. Output of the filtering algorithm applied on input data from Table 7

Duplicate Candidate for Deduplication Non-Duplicate

a. *Humidity* data collection

Measurement *m*	Value *v*	Value Pattern Code	Time stamp *t*		Location stamp *l*				Zone Pattern Code	Combined Pattern Code	Source *s*
			format	*value*	*format*	*value*					
						x	y	z			
Humidity	92 µg/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 µg/m^3	H1	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	Cartesian	4	2	7	Z1	{H1_Z1}	S1
Humidity	103 µg/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 µg/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

Duplicates
Non-Duplicates
Candidates

b. *Temperature* data collection

Measurement *m*	Value *v*	Value Pattern Code	Time stamp *t*		Location stamp *l*				Zone Pattern Code	Combined Pattern Code	Source *s*
			format	*value*	*format*	*value*					
						x	y	z			
Temperature	16 °C	{T1}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	2	5	3	Z2	{T1_Z2}	S1
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}	S1
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	Cartesian	8	6	3	Z4	{T2_Z4}	S1
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	8	6	3	Z4	{T2_Z4}	S1

Non-Duplicates
Candidates
Duplicates

4.5. Fuzzy Redundancy Detection

4.5.1. Fuzzy Inference Agent

The fuzzy redundancy detection module's overall process is shown in Figure 5. 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*).

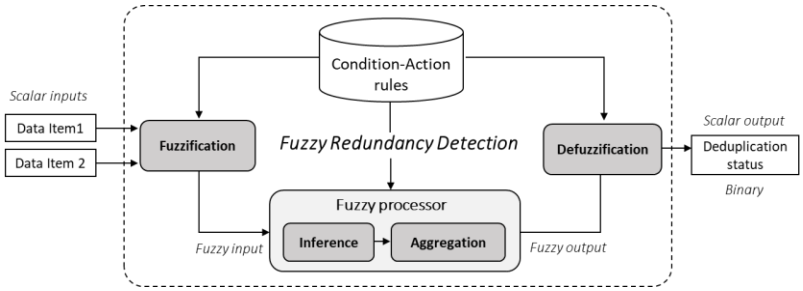
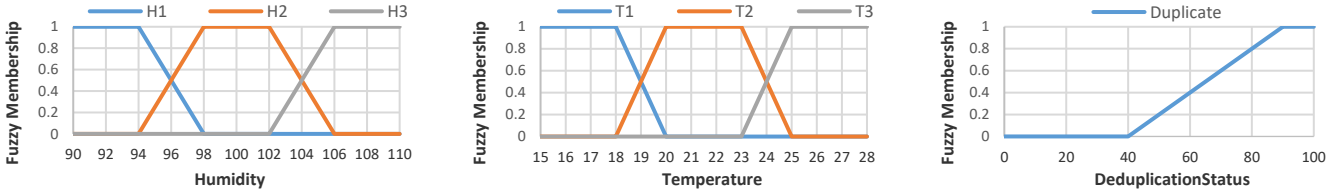


Figure 5. Simplified diagram describing the *fuzzy redundancy detection* module's overall process

Fuzzification: First, the scalar data item values are fuzzified, producing linguistic values associated with fuzzy membership degrees (e.g., *humidity* value 103 µg/m^3 becomes 75% H2 and 25% H3 following Figure 6). 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 6.a and b show the fuzzy partitions for *humidity* and *temperature* measurements which we adopt in our motivating scenario[5]. 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 6.c using one membership function varying from 0-to-100% duplication.



a. Input *humidity* fuzzy partitions, cf. Table 4.a **b.** Input *temperature* fuzzy partitions, cf. Table 4.b **c.** Output *deduplication status* fuzzy partitions

Figure 6. Input *humidity* and *temperature* fuzzy partitions, and output *deduplication status* fuzzy partitions defined using the trapezoidal function following the lookup tables in Table 4

[5] In this example, we adopt trapezoidal functions, yet any other function can be used (e.g., triangular, sinusoidal, or Gaussian) based on domain expert and application needs.

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) (Ross T. J. 2016). In our agent, we adopt *Mamdani's implication* operator as the default inference function given its common usage in the literature (Salloum G. and Tekli J. 2021, Bouchon-Meunier B. *et al.* 2003). 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 (Salloum G. and Tekli J. 2021, Ross T. J. 2016). 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 (Salloum G. and Tekli J. 2021, Ross T. J. 2016). 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:

$$f_1 \Longrightarrow_{\text{Mamdani}} f_2 \equiv f_1 \wedge f_2 \equiv min(f_1, f_2) \quad \textbf{(4)}$$

where $\wedge$ is the AND fuzzy logic operator[6]

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

$$F_{agg} = F_{Max} = max(f_1, f_2, \ldots, f_n) \quad \textbf{(5)}$$

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

$$x = \frac{\int x \times F_{agg}(x) \times dx}{\int F_{agg}(x) \times dx} \quad \textbf{(6)}$$

4.5.2. Computation Example

We consider in Table 8 two cases for *humidity* and *temperature* measurements studied in our motivation scenario. The detailed computation process for *humidity* is described in Figure 7. A similar computation process for *temperature* is provided in the appendix. For the *humidity* case, the agent recommends that input 103 µg/m^3 and 104 µg/m^3 data values are duplicates with a 76% fuzzy membership degree, which seems reasonable given the *humidity* lookup tables and value ranges defined in Table 4 (H2 and H3 fuzzy partitions intersect between [102, 106] µg/m^3, where 103 is much closer to the 102 µg/m^3 boundary of H2 than to the 106 µg/m^3 boundary of H3, but also 103 µg/m^3 and 104 µg/m^3 are close to each other). Similarly 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 (T1 and T2 fuzzy partitions intersect between [18, 20] °C, where 19.5 is closer to the 20 °C

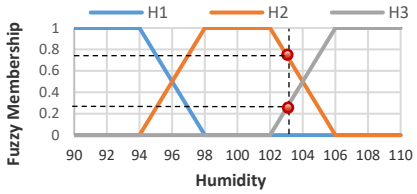
[6] The AND fuzzy logic operator can be any t-norm function, including *min* which is commonly adopted in the literature.

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 (cf. experiments in Section 7).

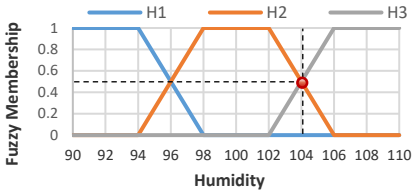
Given our running example data from Table 8, the identified *humidity* and *temporal* redundancies following the fuzzy redundancy detection process are shown in Table 9.

1. Fuzzification: Given *case 1*'s input data: Humidity_DataItem1 = 103 µg/m^3 and Humidity_DataItem2 = 104 µg/m^3, we compute the corresponding fuzzy membership values following the *humidity* fuzzy functions in Figure 6.a (reported below):

- For Humidity_DataItem1:
  $f_{H1}(103) = 0$, $f_{H2}(103) = 0.75$, and $f_{H3}(103) = 0.25$



- For Humidity_DataItem2:
  $f_{H1}(104) = 0$, $f_{H2}(104) = 0.5$, and $f_{H3}(104) = 0.5$



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

- ***Rule 2***: H2(Humidity_DataItem1) ∧ H2(Humidity_DataItem2) ⇒ *Duplicate*(DedupStatus)
- ***Rule 3***: H3(Humidity_DataItem1) ∧ H3(Humidity_DataItem2) ⇒ *Duplicate*(DedupStatus)

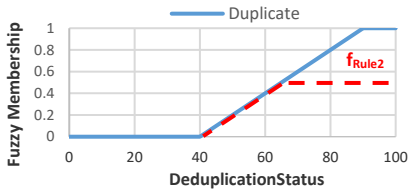
3. Inference: Applying *Mamdani*'s inference mechanism:

Rule 2:
Executing the AND fuzzy operator[5]:

$$min\left(f(103)_{H2}^{Humidity\_DataItem1}, f(104)_{H2}^{Humidity\_DataItem2}\right) = min(0.75, 0.5) = 0.5$$

Executing the implication fuzzy operator:

$$f_{Rule2} = min\left(0.5, Duplicate(\text{DedupStatus})\right)$$

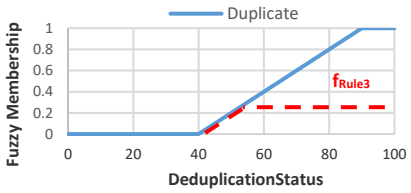


Rule 3:
Executing the AND fuzzy operator:

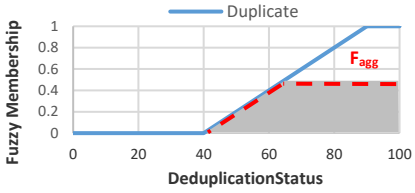
$$min\left(f(103)_{H3}^{Humidity\_DataItem1}, f(104)_{H3}^{Humidity\_DataItem2}\right) = min(0.25, 0.5) = 0.25$$

Executing the implication fuzzy operator:

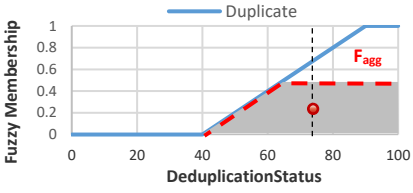
$$f_{Rule3} = min\left(0.25, Duplicate(\text{DedupStatus})\right)$$



4. 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).



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) as the agent's output = 76%.



6. 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% ≥ $dedup_{threshold}$, the agent's final output becomes: dedupStatus = *duplicates*

Figure 7. Fuzzy redundancy detection process for the *humidity* sample case considered in our running example (cf. Table 8)

4.6. Redundancy Removal

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 expert-centric mode which considers domain expert requests that describe the deduplication

requirements/conditions when removing redundancies. Following our running example data from Table 9, the identified *humidity* and *temporal* redundancies can be removed using the auto-removal *median* function in Table 10.

Table 9. Output of the fuzzy redundancy detection process applied on the running example data from Table 8

a. *Humidity* data collection

Duplicate / Non-Duplicate

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Location stamp *l***				**Zone Pattern Code**	**Combined Pattern Code**	**Source *s***	
			format	*value*	*format*	*value* x	*value* y	*value* z				
Humidity	92 µg/m^3	H1	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	Cartesian	1	2	3	Z1	{H1_Z1}	S1	Duplicates
Humidity	94 µg/m^3	H1	dd/MM/yyyy hh:mm:ss	10/02/2019 10:02:00	Cartesian	4	2	7	Z1	{H1_Z1}	S1	
Humidity	103 µg/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	Duplicates
Humidity	104 µg/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	

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Value Pattern Code**	**Time stamp *t***		**Location stamp *l***				**Zone Pattern Code**	**Combined Pattern Code**	**Source *s***	
			format	*value*	*format*	*value* x	*value* y	*value* z				
Temperature	16 °C	{T1}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	2	5	3	Z2	{T1_Z2}	S1	
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}	S1	Duplicates
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:03:00	Cartesian	8	6	3	Z4	{T2_Z4}	S1	
Temperature	21 °C	{T2}	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	8	6	3	Z4	{T2_Z4}	S1	

Note that domain experts 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*), experts can provide their requirements in the form of simple consumer requests that the module translates into redundancy removal rules:

Definition 4 – Consumer Request: We define a consumer request as a 3-tuple:

$$req = \langle c\_id \; ; \; s\_id \; ; \; P \rangle \tag{7}$$

where *c_Id* is the consumer (expert or device) identifier, *s* is the data source (device) identifier, and *P* is a set of consumer preferences for the sensory data produced by the data source. Consumer preferences can be expressed as a 4-tuple:

$$P = \langle target_a \; ; \; freq \; ; \; type \; ; \; f_{rep} \rangle \tag{8}$$

where $target_m$ 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), and 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) ●

Table 10. Output of redundancy removal using the auto-removal *median* function applied on the redundant data collections from Table 9

a. *Humidity* data collection

Measurement *m*	**Value *v***	**Time stamp *t***		**Location stamp *l***				**Source *s***
		format	*value*	*format*	*value* x	*value* y	*value* z	
Humidity	92 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:00:00	Cartesian	1	2	3	S1
Humidity	103 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	5	2	7	S1

b. *Temperature* data collection

Measurement *m*	**Value *v***	**Time stamp *t***		**Location stamp *l***				**Source *s***
		format	*value*	*format*	*value* x	*value* y	*value* z	
Temperature	16 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	2	5	3	S1
Temperature	21 °C	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:00	Cartesian	8	6	3	S1

Figure 8.a shows a sample consumer request where the expert requires a specific deduplication ratio from a given device. Figure 8.b shows the deduplicated data based on the consumer request, considering the running example data from Table 9.

```
<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>
```

a. Sample consumer request

Measurement *m*	Value *v*	Time stamp *t*		Location stamp *l*				Source *s*
		format	*value*	*format*	*value*			
					x	y	z	
Humidity	93 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:01:00	Cartesian	2.5	2	5	S1
Humidity	103.5 µg/m^3	dd/MM/yyyy hh:mm:ss	10/02/2019 10:05:30	Cartesian	6	2	7	S1

b. Redundancy-free data based on the consumer request

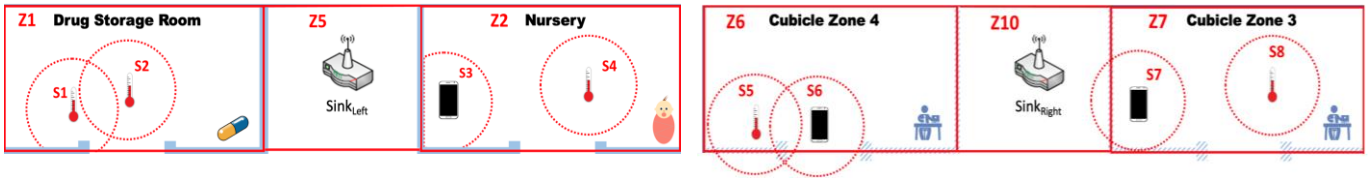
Figure 8. Sample consumer request, and the resulting redundancy removal based on the running example data from Table 9

5. FREDD Sink-Level Deduplication

As mentioned in the motivation of our study, deduplication in connected environments should/can be evaluated on different levels of the network, namely: at the edge level (described in Section 4) and at the sink level, and needs to be adapted for static and mobile edge devices with different configurations of hard-separated or soft-separated zones and coverage areas. In this section, we describe the different sink-level deduplication use cases and explain how FREDD handles every case.

5.1. 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 9). 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 9.a). 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 9.b).



a. Hard-zone separations

b. Soft-zone separations

Figure 9. 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 11: 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 11. Sink-level deduplication uses cases

	Considers Sink Zones	Considers Sensor Coverage Areas	Hard Separations between Zones	Soft Separations between zones
Case 1	✓	×	✓	×
Case 2	✓	✓	✓	×
Case 3	✓	×	×	✓
Case 4	✓	✓	×	✓

Note that in this study, we focus on deduplication within individual sink zones, where each sink comprises an independent system and there is no deduplication or coordination required between the sinks. Yet, we can extend our scenario to consider inter-sink collaborations for handling redundancies that occur in overlapping sink zones. This would add a new layer of deduplication handling within the connected network environment, which we plan to investigate in a dedicated future study.

5.1.1. 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 9.a 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.

5.1.2. 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 9.a. 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 9.a). 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 5.2).

5.1.3. 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 9.b). Deciding on coverage area-zone inclusion[7] and overlapping is done by evaluating the spatial topological relations between the areas and the zones (similarly to area-area topological relations, cf. Section 5.2).

5.1.4. 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 9.b). 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 9.b) will not be considered for deduplication at the sink node.

5.2. Sink-Level Deduplication Process

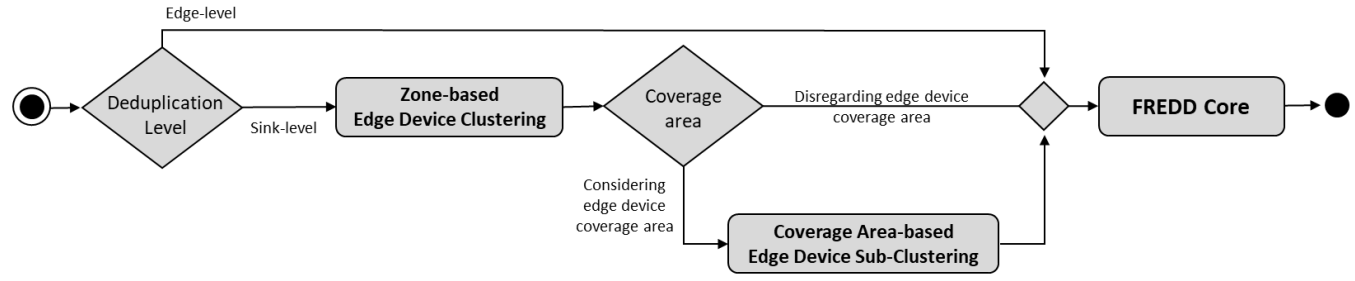


Figure 10. FREDD's sink-level deduplication process

The overall process of sink-level deduplication using FREDD is depicted in Figure 10. 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 (Abebe M. *et al.* 2020, Taddesse F.G. *et al.* 2009):

[7] Note that *inclusion* is an asymmetric *equality* relation which is evaluates based on an asymmetric similarity measure (i.e., area-zone *inclusion* is evaluated using $Sim_{ASum}(area, zone) = \frac{|\,Intersection(area, zone)\,|}{|\,area\,|}$, where the relation occurs if $Sim_{Asym}(area, zone) \geq Thresh_{Include}$ e.g., (Abebe M. et al. 2020, Taddesse F.G. et al. 2009).

$$Sim(area, zone) = \frac{|\, Intersection(area, zone)\,|}{min(|\, area\,|, |\, zone\,|)} \quad \textbf{(9.1)}$$

$$Sim(area_1, zone_2) = \frac{|\, Intersection(area_1, area_2)\,|}{min(|\, area_1\,|, |\, area_2\,|)} \quad \textbf{(9.2)}$$

The amount of overlapping between an area and a zone is controlled by the expert through a dedicated similarity threshold (cf. Figure 11.a, 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 (cf. Formulas 9.1 & 9.2) **b.** Using an asymmetric similarity measure (cf. Formula 10)

Figure 11. Basic spatial topological relationships following (Abebe M. *et al.* 2020, Taddesse F.G. *et al.* 2009)

Similarly, the *inclusion* relation between and area and a zone is evaluated using an asymmetric version of the geometric similarity measure, more formally (Abebe M. *et al.* 2020, Ehrig M. and Sure Y. 2004):

$$Sim_{Asym}(area, zone) = \frac{|\, Intersection(area, zone)\,|}{|\, area\,|} \quad \textbf{(10)}$$

where the *inclusion* relation occurs if $Sim_{Asym}(area, zone) \geq Thresh_{Include}$ (cf. Figure 11.b).

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 9.2) 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 11.a, e.g., $Sim(area_1, area_2) \geq Thresh_{Overlap}$ then coverage areas largely overlap and their edge devices are eligible for sink-level deduplication).

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.

6. Complexity Analysis

The overall time complexity of our FREDD approach simplifies to: $O(N \times E^2)$ 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 edge-level (and sink-level only) deduplication: the complexities of FREDD's modules are linear w.r.t. the number of data items being processed at the edge (or at the sink), and simplify in the worst case scenario to $O(N) + \ldots + O(N)$, which comes down to an overall $O(N)$.
- For edge-and-sink level deduplication: we facture-in: i) the number of edge devices E per sink node, where the system requires worst case $O(((E \times (E\text{-}1))/2)$, and simplifies to $O(E^2)$ 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 $O(E \times S)$ 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 $O(N \times (E^2 + E \times S))$ which simplifies to $O(N \times E^2)$ since E is generally much larger than S.

7. Experimental Evaluation

We have implemented our *FREDD* framework as a web-based application, using methods from the *jFuzzyLogic* open source library (Cingolani P. and Alcalá-Fdez J. 2013, Cingolani P. and Alcala-Fdez J. 2012) in implementing our fuzzy logic agent, to allow easy manipulation for domain experts in operating and evaluating the system[8]. 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[9].

[8] 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.

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

7.1. Experimental Test Data

We build three datasets for edge device, mobile device, and sink device measurements collected from the Intel Lab Berkeley dataset (Bodik P. *et al.* 2019) obtained from 54 Micra2Dot sensors depicted in the Figure 12. 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:

- **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 → S2 → … → S9 → S10 → S9 → S8…→ S1), iii) mobile sensor M1 collects from each location a random number of data measurements[10], 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.
- **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.

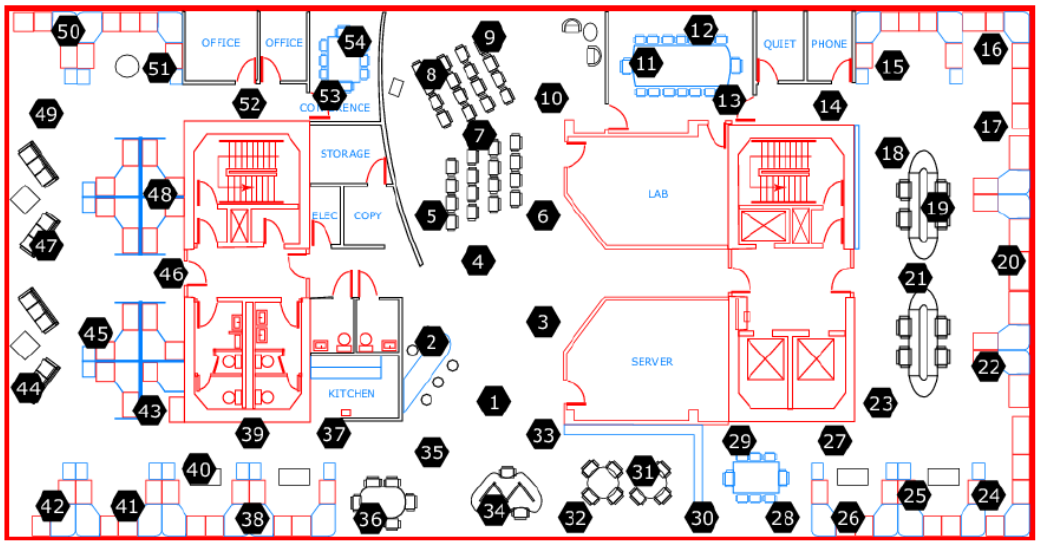


Figure 12. Schematic of the Intel Lab Berkeley Micra2Dot sensors

7.2. 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; and at the sink device, we add two more metrics: iii) size of transmitted data, and iv) size of stored data. We describe the four 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 (Ismael W. *et al.* 2019). More formally, given $TS_o = [(t_1, v_{o\_1}), (t_2, v_{o\_2}), ..., (t_n, v_{o\_n})]$ 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. 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

[10] 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.

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:

$$acc = \frac{\sum_{i=1}^{n} min(v_{o\_i}, v_{r\_i})}{\sum_{i=1}^{n} max(v_{o\_i}, v_{r\_i})} \in [0,1] \quad \textbf{(11)}$$

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:

$$redu = \left(1 - \frac{|TS_o| - |TS_d|}{|TS_o|}\right) \in [0,1] \quad \textbf{(12)}$$

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.

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[11]. 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.

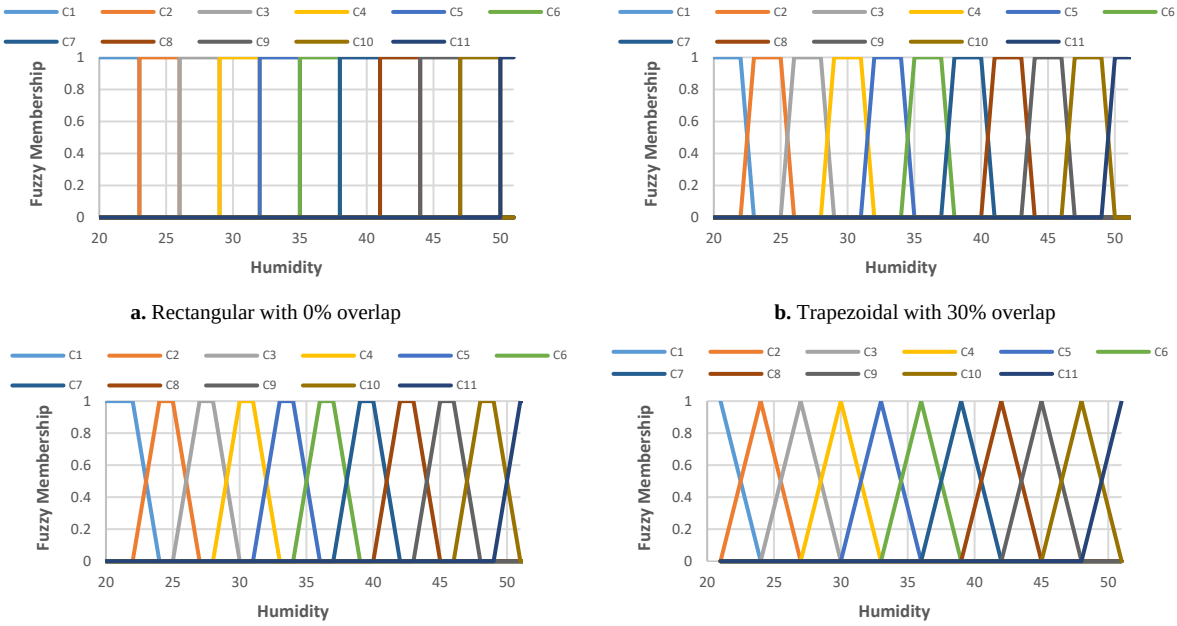
7.3. Quality Evaluation

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, ii) fuzzy deduplication threshold, iii) sink zone granularity, iv) number of edge devices connected to the sink, and v) sensor coverage area size. We also conduct a vi) baseline comparison evaluating FREDD's deduplication quality compared with its most recent alternatives. We describe the experiments and their results in the below sub-sections.

7.3.1. 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* from our motivation scenario). We consider four different pattern code fuzzy membership functions as shown in Figure 13: 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 14. 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).

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



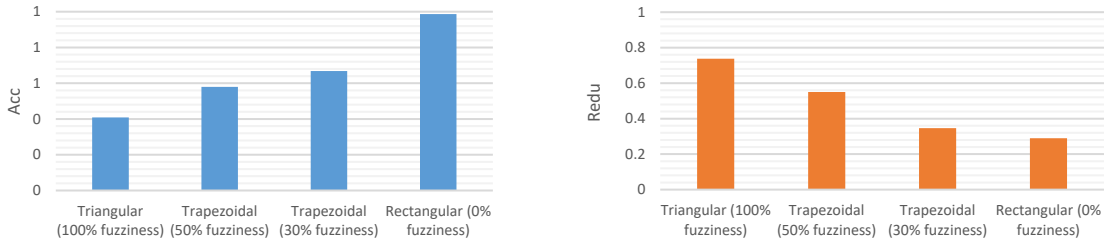
a. Rectangular with 0% overlap

b. Trapezoidal with 30% overlap

c. Trapezoidal with 50% overlap

d. Triangular with 100% overlap

Figure 13. Different *humidity* pattern code fuzzy membership functions with different boundary range overlapping sizes



a. Deduplication accuracy (*acc*)

b. Data reduction ratio (*redu*)

Figure 14. *Acc* and *redu* results when varying the overlapping size of boundary data ranges

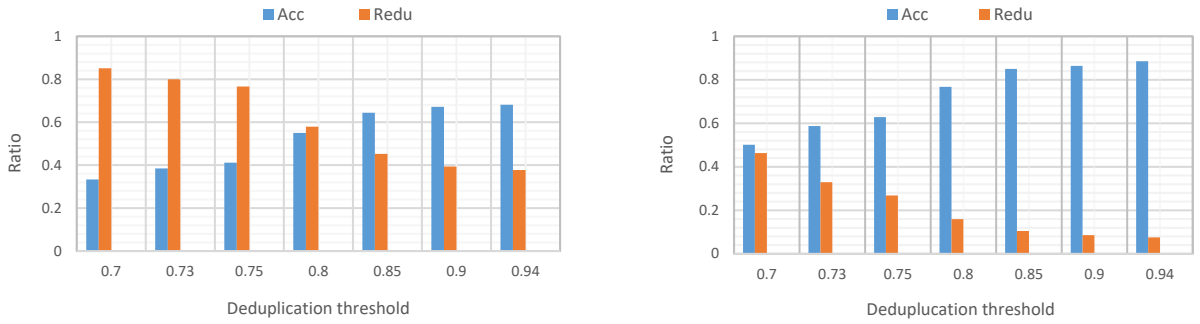
7.3.2. Fuzzy Deduplication Threshold evaluation

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. *challenge 1* from our motivation scenario), and consider the impact of device mobility on the deduplication process (cf. *challenge 2*). 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*). Figures 15 and 16 show the results of deduplication quality metrics when applied on static edge data from *dataset 1* (Figure 15.a), and on mobile edge data from *dataset 2* (Figure 15.b), 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. We notice a similar behavior in Figure 16, which shows the results of deduplication quality metrics considering sink device data from *dataset 3*. Here, we highlight the following observations:

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

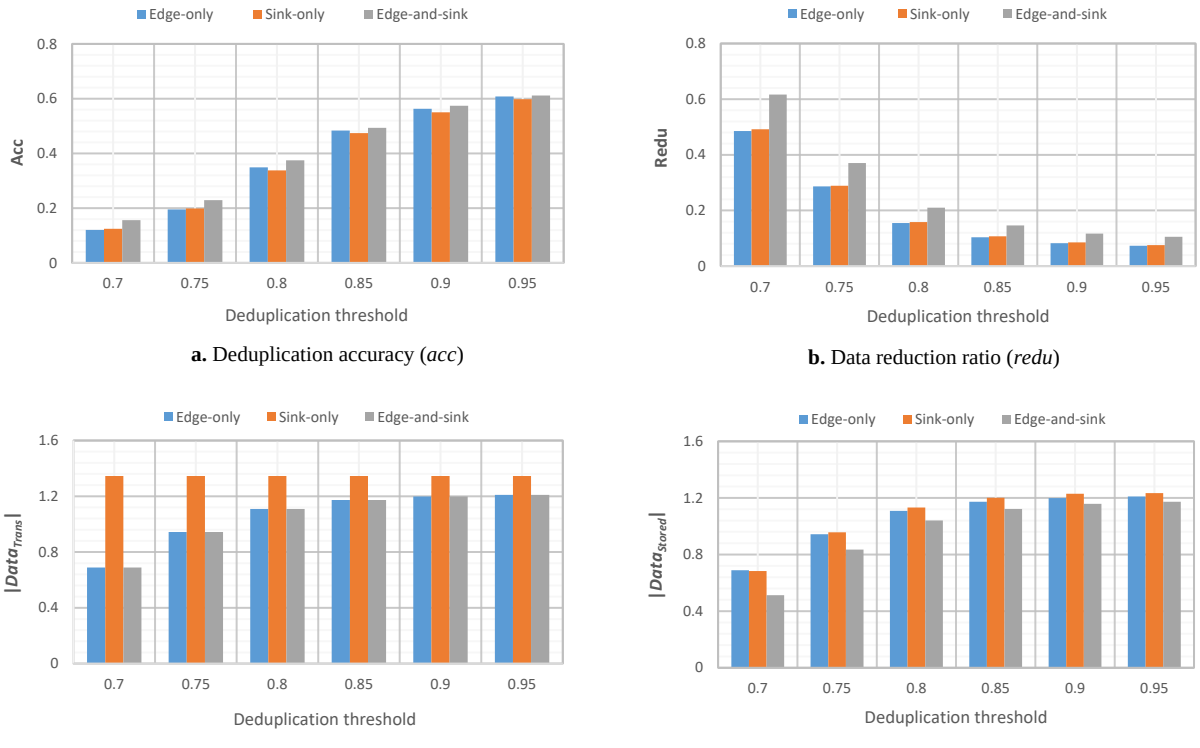
- $|data_{Stored}|$ is smallest in case of deduplication at edge-and-sink, compared with edge-only and sink-only, since deduplication is performed at two levels.

Note that fine-tuning and optimizing the evaluation metric values can be handled automatically as a multi-objective optimization problem. This can be solved using a number of known techniques that apply linear programming and machine learning to identify the best weights for a given problem class, e.g., (Zou F. *et al.* 2021, Gal A. *et al.* 2016, Hopfield J. J. 1989). We do not further discuss evaluation metric optimization here since it is out of the scope of this paper, and we report it to a dedicated future study.



a. Resuls with static device data from dataset 1 **b.** Resuls with mobile device data from dataset 2

Figure 15. *Acc* and *redu* results with varying fuzzy deduplication thresholds



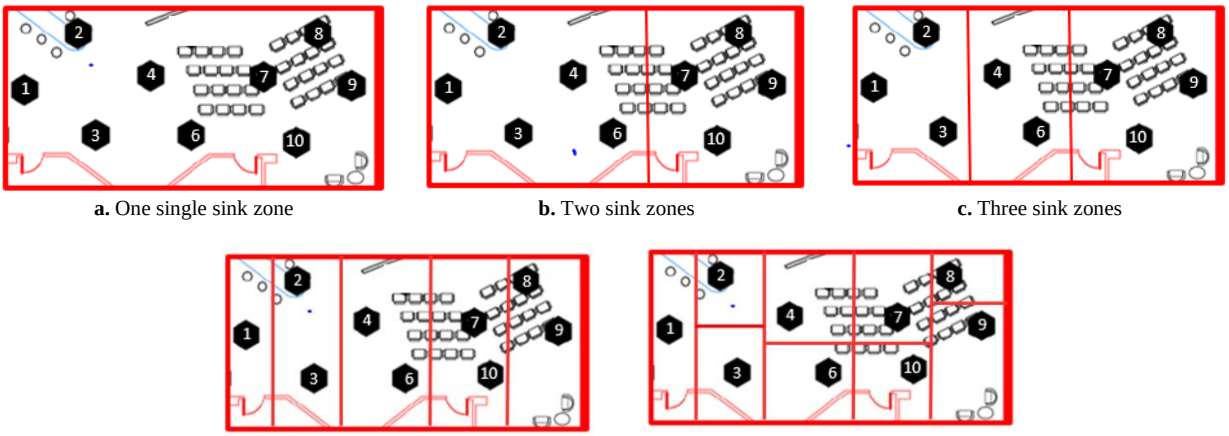
a. Deduplication accuracy (*acc*) **b.** Data reduction ratio (*redu*)

c. Size of data transmitted to the sink ($|data_{Trans}|$) **d.** Size of data stored at the sink ($|data_{Trans}|$)

Figure 16. Deduplication quality metrics obtained with varying fuzzy deduplication thresholds

7.3.3. 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 17. We first consider the area shown in Figure 17.a 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 17.b-to-f). We perform this experiment considering deduplication at the edge and sink levels (cf. *challenge 1* of our motivation scenario), and deduplication with static and mobile devices (cf. *challenge 2*). 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*).



a. One single sink zone **b.** Two sink zones **c.** Three sink zones

e. Five sink zones **f.** Nine (individual device) sink zones

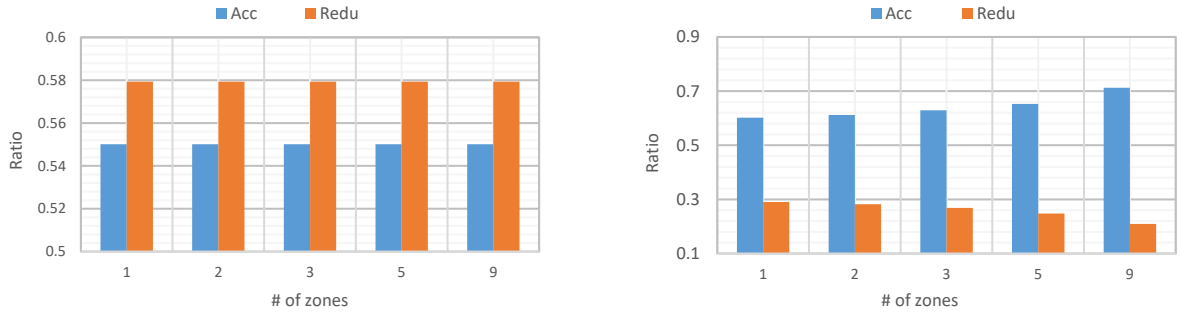
Figure 17. Sample sink zone granularities using an extract of our reference Micra2Dot dataset (cf. Figure 12)

Figures 18 and 19 show the results of deduplication quality metrics when applied on static edge data from *dataset 1* (Figure 18), and on mobile edge data from *dataset 2* (Figure 19), 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.

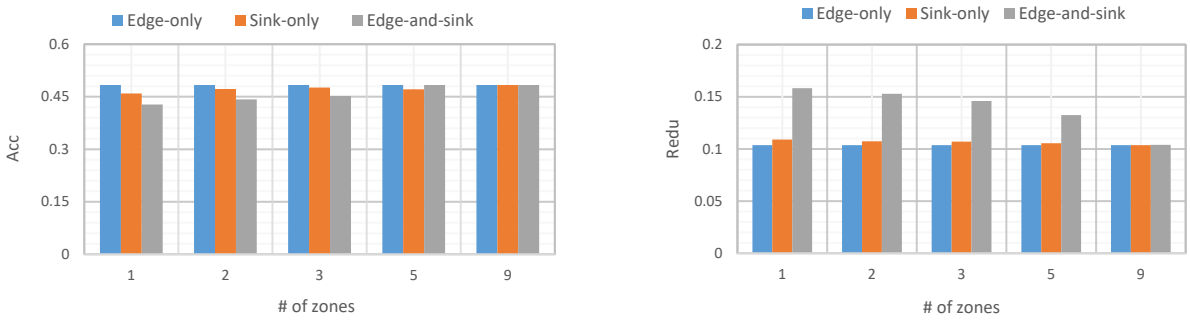
We notice a similar behavior in Figure 19 which shows the results of deduplication quality metrics considering sink device data from *dataset 3*. Here, we highlight the following observations:

- 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.



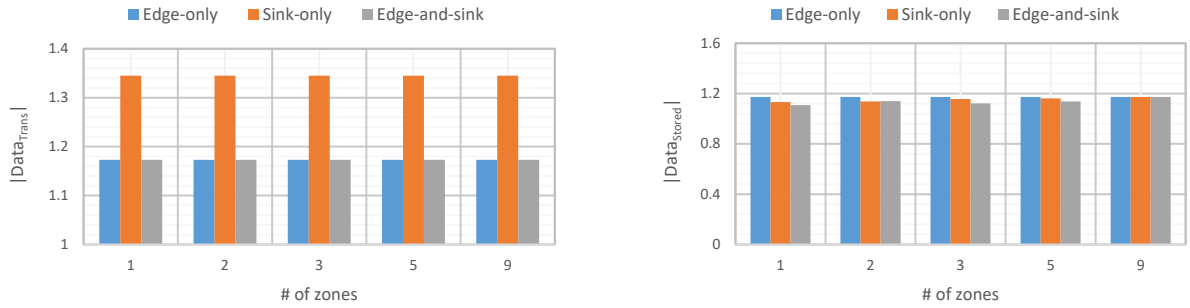
a. Resuls with static device data from dataset 1 **b.** Resuls with mobile device data from dataset 2

Figure 18. *Acc* and *redu* results with varying zone granularities considering static edge nodes



a. Deduplication accuracy (*acc*)

b. Data reduction ratio (*redu*)



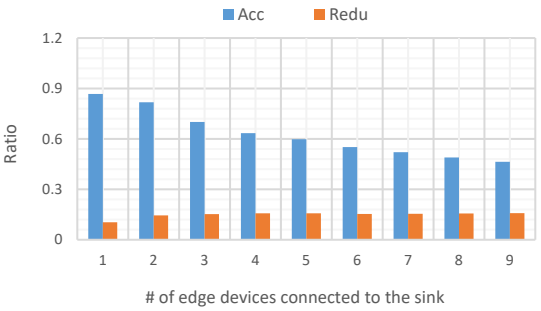
c. Size of data transmitted to the sink ($|data_{Trans}|$)

d. Size of data stored at the sink ($|data_{Stored}|$)

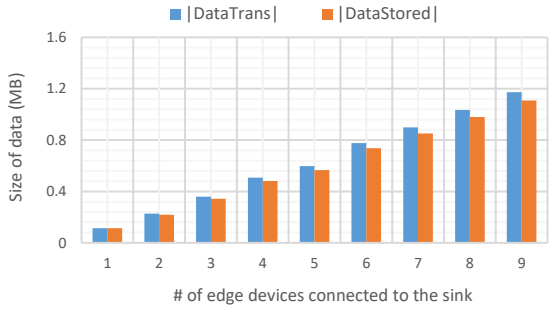
Figure 19. Deduplication quality metrics obtained with varying zone granularities considering mobile edge nodes

7.3.4. 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*). Figure 20 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 20.a 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 20.b 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*)

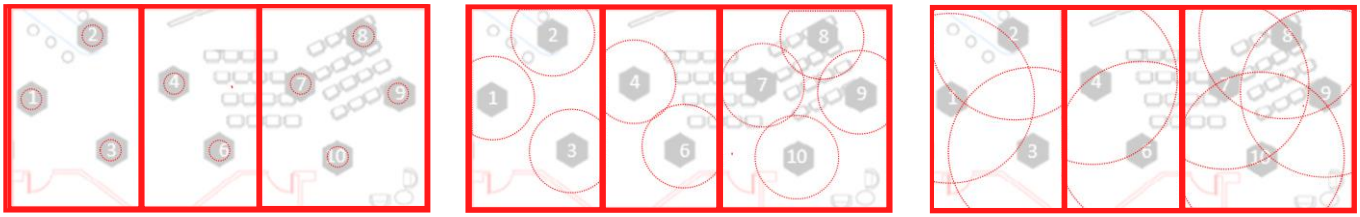


b. Size of data transmitted ($|data_{Trans}|$) and stored ($|data_{Stored}|$) at the sink

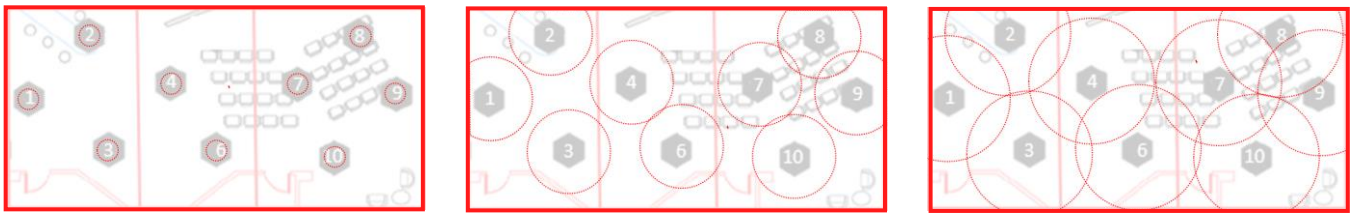
Figure 20. Deduplication quality metrics applied on sink node data (*dataset 3*), when varying the number of devices per sink zone

7.3.5. Sensor Coverage Area Size evaluation

In this experiment, we evaluate the impact of varying the radius of sensor coverage areas on the deduplication process (cf. *challenge 3*). We consider in Figure 21 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. Illustrations of hard separations between zones with coverage area radius = 1, 4, and 6 respectively

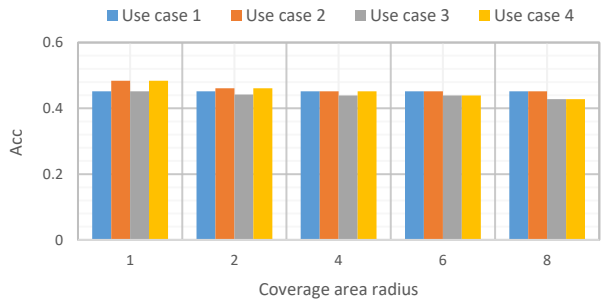


b. Illustrations of soft separations between zones use with coverage area radius = 1, 4, and 6 respectively

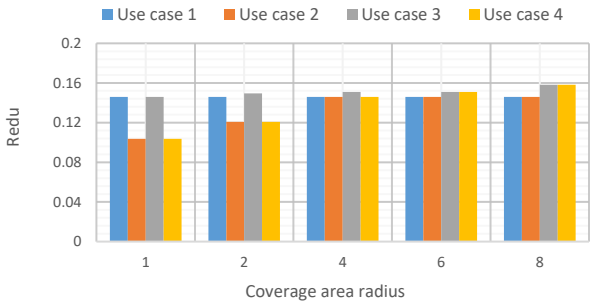
Figure 21. Sample sink zone use cases with sensor coverage area radius variations using an extract of the Micra2Dot dataset

Figure 22 shows deduplication quality metrics when applied on sink node data from *dataset 3*. Here, we highlight the following observations:

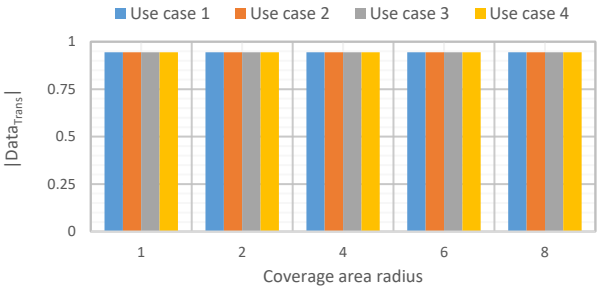
- In case 1 – *zone-based with hard separations* (Figure 22.a): *acc*, *redu*, and $|data_{stored}|$ are not affected by the size of the sensor's coverage area, since the deduplication process is only affected by the location of the sensor, and not its coverage area. Note that $|data_{trans}|$ in unaffected in all cases 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 22.a) and in case 4 – *zone-and-coverage based with soft separations* (cf. Figure 22.b), 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. 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).



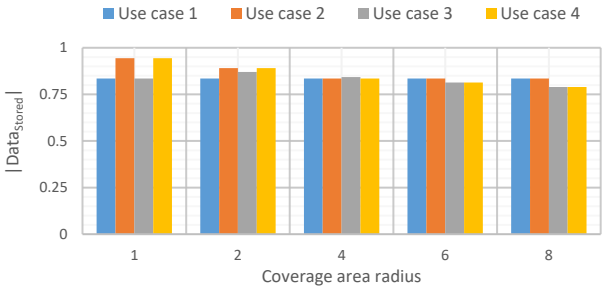
a. Deduplication accuracy (*acc*)



b. Data reduction ratio (*redu*)



c. Size of data transmitted to the sink ($|data_{Trans}|$)



d. Size of data stored at the sink ($|data_{Stored}|$)

Figure 22. Deduplication quality metrics applied on sink node data (*dataset 3*), when varying sensor coverage area radius with multiple hard/soft separation use cases

- In case 3 – *zone-based with soft separations* (Figure 22.b), *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.

7.3.6. Baseline Comparison with Existing Approaches

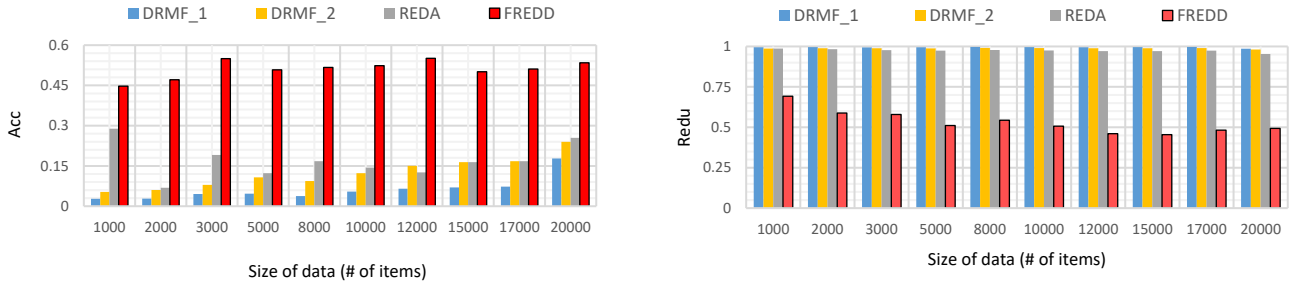
In order to further evaluate our solution, we conducted a comparative study to assess its effectiveness with respect to recent alternatives in the literature. On the one hand, our solution i) handles redundancies at both edge device and sink device levels of the network (cf. *challenge 1* of our motivation scenario), 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 (Mansour E. *et al.* 2020) (cf. comparative Table 12). 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 12. Comparing FREDD with alternative solutions

	Challenge 1		Challenge 2			Challenge 3
	Edge-level deduplication	Sink-level deduplication	Device mobility	Zone separations (hard/soft)	Sensor coverage area size	Domain expert control
SVM (Patil P. and Kulkarni U. 2013)	×	✓	×	×	×	×
CWCA (Ullah A. et al. 2019)	×	✓	×	×	×	✓
REDA (Khriji S. *et al.* 2018)	✓	×	×	×	×	×
DRMF (Mansour E. *et al.* 2020)	✓	×	✓	×	×	×
FREDD (our solution)	✓	✓	✓	✓	✓	✓

We experimentally compare our method's effectiveness with two of its most recent alternatives: i.e., REDA (Khriji S. *et al.* 2018) and DRMF (Mansour E. *et al.* 2020). To test REDA, we consider the crisp humidity ranges shown in Figure 13.a. To test FREDD, we consider the fuzzy humidity ranges in Figure 13.b 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 23 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 and DRMF1/2. 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 23. Comparison of the deduplication quality metrics between RED, DRMF1/2 and FREDD, when varying the number of data measurements of *dataset1*

To further explain the results in Figure 23, 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 24 shows the percentage of deduplication produced by each algorithm for a first humidity value of 39.5 µg/m^3, and the second value with a variation range of ± 2.5 µg/m^3.

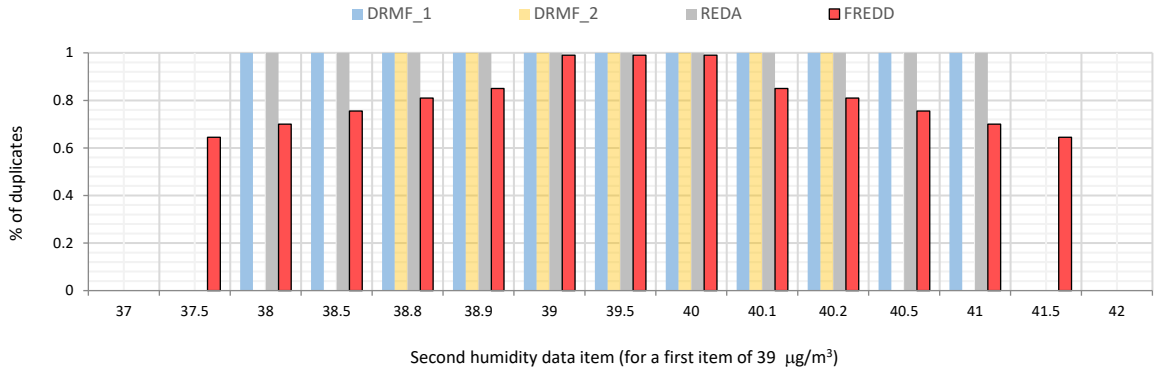
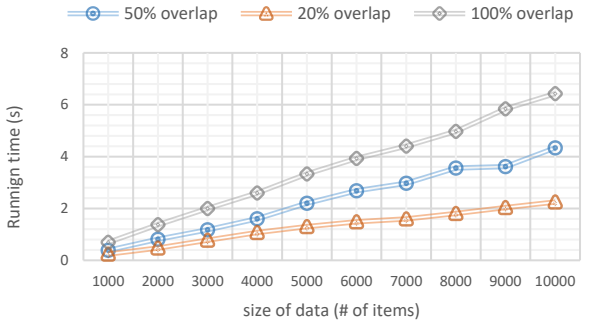
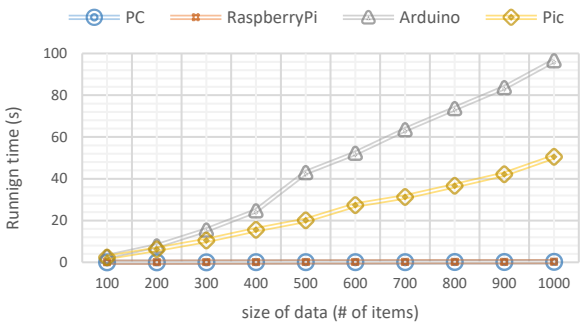


Figure 24. Percentage of deduplicates when fixing the first humidity data to 39.5 µg/m^3 and varying the second between [37, 42] µg/m^3

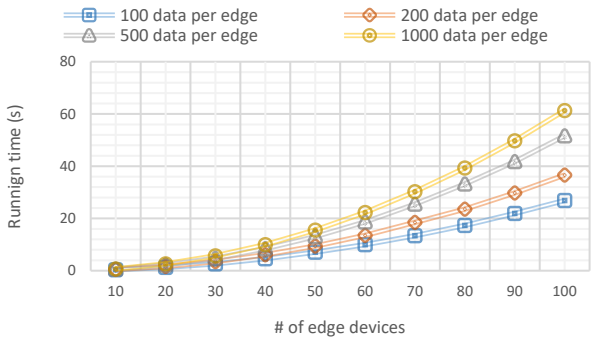
Considering REDA's results, all values that lie between [38, 41] µg/m^3 are considered automatic duplicates (i.e., 100% duplicates) and are assigned the same pattern code. With DRMF1, considering a delta δ=2/3 and given a cluster centroid of 39.5 µg/m^3, all values that lie between [38, 41] µg/m^3 are 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 24. A similar behavior is also noticed for DRMF2. 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] µg/m^3), and ii) a fuzzy range (i.e., between [37, 39] µg/m^3 and [40, 42] µg/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 µg/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 24).



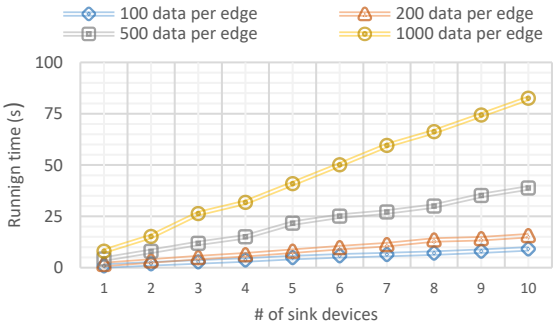
a. Edge-level processing time when varying the number of data items



b. Edge-level processing time on different hardware devices, considering 50% data item overlap



c. Edge-and-sink level processing time when varying the number of edge devices per sink note



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

Figure 25. FREDD's time performance consider edge-level and edge-and-sink level deduplication processes with varying parameters

7.4. 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. Tests were carried out on a PC with an *Intel I7* system with 2.9 GHz CPU/16GB RAM. Edge-level computations were also carried out on *Arduino Uno*, *PIC32MX* and *Raspberry PI* devices, with 16 MHz/2KB RAM, 50 MHz/32KB RAM and 1.5GHz/4GB RAM respectively. Results in Figure 25.a and b highlight the linear complexity of FREDD's edge-based deduplication process when varying the number of data items per edge node, reflecting $O(N)$ time complexity. Results in Figure 25.b reflect the linear computation overhead added by less performing hardware edge devises like *PIC32MX* and *Arduino Uno*, compared with the more powerful *Raspberry PI*. Results in Figure 25.c highlight the polynomial complexity of FREDD's edge-and-sink deduplication process when varying the number of edge devices per sink node, reflecting $O(E^2)$ time. Figure 25.d highlights FREDD's its linear complexity when varying the number of sink nodes, reflecting $O(E \times S)$ time.

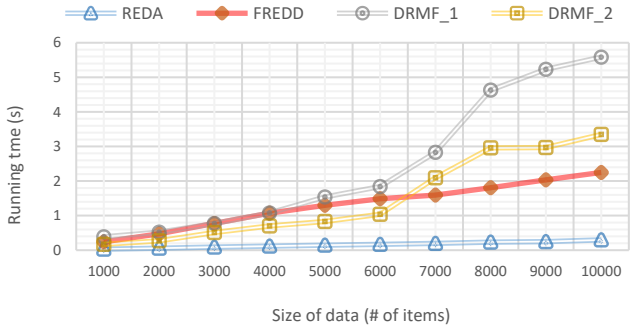


Figure 26. FREDD's time performance compared with its recent alternatives, considering a fixed data size of 1000 items per edge, and a fixed number of 10 edge devices per sink node

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 26 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.

8. Conclusion

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 general network infrastructure including both the edge level and the sink level of the network. Moreover, it is adapted for multiple scenarios, considering both static and mobile devices, with different configurations of hard-separated and soft-separated zones, and different sensor coverage areas. Experiments on a real-world dataset highlight FREDD's potential and improvement compared with existing solutions.

We are currently investigating the use of parametric learners (Wen X. 2021, Abboud R. and Tekli J. 2019) and meta-heuristic algorithms (Nguyen T. 2021, Azar D. *et al.* 2016) 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 (Jebbaoui H. *et al.* 2015) 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 (Haraty R. and El Sai M. 2017, Haraty R. *et al.* 2016) in connected environments, including damage assessment and recovery from deduplicated data.

Declarations

Conflicts of interest/Competing interests
Not Applicable.

Compliance with Ethical Standards

- Disclosure of potential conflicts of interest
  1. Conflict of Interest: *The authors declare that they have no conflict of interest.*

- Research involving human participants and/or animals
  1. Statement of human rights: *Ethical approval: For this type of study formal consent is not required.*
  2. Statement on the Welfare of Animals: *Ethical approval: This article does not contain any studies with animals performed by any of the authors.*

- Informed consent: *Additional informed consent was obtained from all individual participants for whom identifying information is included in this article.*

Availability of Data

The experimental data and results are available at the following link: http://sigappfr.acm.org/Projects/FREDD/. Additional information can be acquired the authors on reasonable request.

Availability of Code

An executable version of the prototype are available at the following link: http://sigappfr.acm.org/Projects/FREDD/.

References

Abboud R. and Tekli J. (2019). *Integration of Non-Parametric Fuzzy Classification with an Evolutionary-Developmental Framework to perform Music Sentiment-based Analysis and Composition.* Springer Soft Computing 24(13): 9875-9925

Abebe M., Tekli J., Getahun F., Chbeir R. and Tekli G. (2020). *Generic Metadata Representation Framework for Social-based Event Detection, Description, and Linkage.* Knowledge Based Systems 188.

Attigeri G., Karunakar A. and Maddodi S. (2010). *Data Deduplication Techniques and Analysis.* , International Conference on Emerging Trends in Engineering & Technology pp. 664-668.

Azar D., Fayad K. and Daoud C. (2016). *A Combined Ant Colony Optimization and Simulated Annealing Algorithm to Assess Stability and Fault-Proneness of Classes Based on Internal Software Quality Attributes.* International Journal of Artificial Intelligence (ISSN 0974-0635) 14:2.

Bhalerao A. and Pawar A. (2017). *A Survey on Data Deduplication for Efficiently Utilizing Cloud Storage for Big Data Backups.* International Conference on Trends in Electronics and Informatics (ICEI) pp. 933-938.

Bodik P., Hong W., Guestrin C., Madden S., Paskin M. and Thibaux R. (2019). *Intel Lab Data.* Available online: http://db.csail.mit.edu/labdata/labdata.html (accessed on January 2022).

Bouchon-Meunier B., Mesiar R., Marsala C. and Rifqi M. (2003). *Compositional Rule of Inference as an Analogical Scheme.* Fuzzy Sets and Systems 138(1): 53-65.

Chowdhury S. and Benslimane A. (2018). *Relocating Redundant Sensors in Randomly Deployed Wireless Sensor Networks.* IEEE Global Communications Conference (GLOBECOM) pp. 1-6.

Christen P. (2012). *A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication.* IEEE Transactions on Knowledge and Data Engineering 24(9): 1537-1555.

Cingolani P. and Alcala-Fdez J. (2012). *jFuzzyLogic: a Robust and Flexible Fuzzy-Logic Inference System Language Implementation.* In IEEE International Conference on Fuzzy Systems pp. 1-8.

Cingolani P. and Alcalá-Fdez J. (2013). *jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming.* Int. J. Comput. Intell. Syst. 6(1): 61–75.

Ehrig M. and Sure Y. (2004). *Ontology Mapping - an Integrated Approach.* Proceedings of the European Semantic Web Conference (ESWC) pp. 76-91. Heraklion, Greece.

Gal A., Roitman H. and Sagi T. (2016). *From Diversity-based Prediction to Better Ontology & Schema Matching.* Inter. WWW Conference pp. 1145-1155.

Haraty R. and El Sai M. (2017). *Information Warfare: a Lightweight Matrix-based Approach for Database Recovery.* Knowlegde and Information Systems 50(1): 287-313 (2017).

Haraty R., Zbib M. and Masud M. (2016). *Data Damage Assessment and Recovery Algorithm from Malicious Attacks in Healthcare Data Sharing Systems.* Peer Peer Netwwork Applications 9(5): 812-823 (2016).

Hopfield J. J. (1989). *The Effectiveness of Neural Computing.* IFIP World Computer Congress (WCC'89) 402-409.

IoT Analytics (2021). *State of IoT 2021.* https://iot-analytics.com/number-connected-iot-devices/ (accessed January 2022).

Ismael W., Gao M., Al-Shargabi A. and Zahary A. (2019). *An In-Networking Double-Layered Data Reduction for Internet of Things (IoT).* Sensors 19(4): 795.

Jebbaoui H., Mourad A., Otrok H. and Haraty R. (2015). *Semantics-based Approach for Detecting Flaws, Conflicts and Redundancies in XACML Policies.* Computers & Electrical Engineering journal 44: 91-103 (2015).

Kaur R., Chana I. and Bhattacharya J. (2018). *Data Deduplication Techniques for Efficient Cloud Storage Management: a Systematic Review*. Journal of Supercomputing 74(5): 2035-2085.

Khriji S., Raventos G. V., Kammoun I. and Kanoun O. (2018). *Redundancy Elimination for Data Aggregation in Wireless Sensor Networks.* International Multi-Conference on Systems, Signals & Devices (SSD'18) 2018: 28-33.

Li D., Cai Z., Deng L. and Yao X. (2019). *IoT Complex Communication Architecture for Smart Cities based on Soft Computing Models.* Soft Computing 23(8): 2799-2812.

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

Liansheng T. and Wu M. (2015). *Data Reduction in Wireless Sensor Networks: A Hierarchical LMS Prediction Approach.* IEEE Sensors Journal 16.6 (2015): 1708-1715.

Lytras M., Al-Halabi W., Zhang J., Masud M. and Haraty R. (2015). *Enabling Technologies and Business Infrastructures for Next Generation Social Media: Big Data, Cloud Computing, Internet of Things and Virtual Reality.* The Journal of Universal Computer Science 21(11): 1379-1384.

Malhotra J. and Bakal J. (2015). *A Survey and Comparative Study of Data Deduplication Techniques.* International Conference on Pervasive Computing (ICPC) pp. 1-5.

Mansour E., Shahzad F., Tekli J. and Chbeir R. (2020). *Data Redundancy Management in Connected Environments.* International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM-Q2SWinet) pp. 75-80.

Mortadha H. and Jihad A. (2015). *Using Fuzzy Logic Technique to Eliminate the Duplicates in Large Database.* Journal of University of Human Development 1(423) :423-426.

Murtadha H. and Sami S. (2016). *Using Q-Gram and Fuzzy Logic Algorithms for Eliminating Data Warehouse Duplications.* International Arab Conference on Information Technology (ACIT'2016) 8 p.

Nguyen T. (2021). *A Novel Metaheuristic Method based on Artificial Ecosystem-based Optimization for Optimization of Network Reconfiguration to Reduce Power Loss.* Soft Computing 25(23): 14729-14740.

Nižetić S. et al. (2020). *Internet of Things (IoT): Opportunities, Issues and Challenges towards a Smart and Sustainable Future.* Journal of Cleaner Production 274: 122877.

Papageorgiou A., Cheng B. and Kovacs E. (2015). *Real-Time Data Reduction at the Network Edge of Internet-of-Things Systems.* Conference on Network and Service Management (CNSM) pp. 284-291.

Patil P. and Kulkarni U. (2013). *SVM-based Data Redundancy Elimination for Data Aggregation in Wireless Sensor Networks.* Inter. IEEE Conference on Advances in Computing, Communications and Informatics (ICACCI) pp. 1309–1316.

Paulo J. and Pereira J. (2014). *A Survey and Classification of Storage Deduplication Systems.* ACM Computing Surveys 47(1): 11:1-11:30.

Qutub B., Kia M. and Niki P. (2012). *Data Reduction in Low Powered Wireless Sensor Networks.* Wireless Sensor Networks- Technology and Applications 10.5772/50178.

Ross T. J. (2016). *Fuzzy Logic with Engineering Applications.* Wiley; 4th edition 580 p.

Salloum G. and Tekli J. (2021). *Automated and Personalized Nutrition Health Assessment, Recommendation, and Progress Evaluation using Fuzzy Reasoning.* International Journal of Human-Computer Studies (IJHCS) Volume 151, 102610.

Santini S. and Romer K. (2006). *An Adaptive Strategy for Quality-based Data Reduction in Wireless Sensor Networks.* 3rd international Conference on Networked Sensing Systems (INSS'06) 14407470.

Shahri H. and Barforush A. (2004). *Data Mining for Removing Fuzzy Duplicates using Fuzzy Inference.* IEEE Annual Meeting of the Fuzzy Information (NAFIPS) 10.1109/NAFIPS.2004.1336319.

Taddesse F.G., Tekli J., Chbeir R., Viviani M. and Yétongnon K. (2009). *Relating RSS News/Items.* Proceedings of the 9th International Conference on Web Engineering (ICWE'09), LNCS pp. 44-452, San Sebastian, Spain.

Ullah A. et al. (2019). *Secure Healthcare Data Aggregation and Deduplication Scheme for FoG-Orineted IoT.* IEEE International Conference on Smart Internet of Things (SmartIoT) pp. 314–319.

Vlachos I. K. and Sergiadis G. D. (2007). *Intuitionistic Fuzzy Information - Applications to Pattern Recognition.* Pattern Recognition Letters 28(2): 197-206.

VoucherCloud (2018). *The Uses of, and Science Behind, Big Data.* https://www.vouchercloud.com/resources/everyday-big-data (accessed Jamuary 2022).

Wen X. (2021). *Using Deep Learning Approach and IoT Architecture to Build the intelligent Music Recommendation System.* Soft Computing 25(4): 3087-3096.

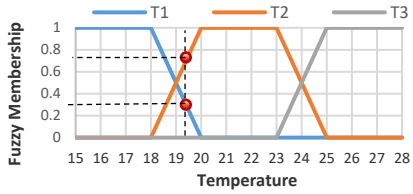
Zadeh L. A. (1984). *Making Computers Think Like People.* IEEE. Spectrum 8:26-32.

Zou F., Yen G., Tang L. and Wang C. (2021). *A Reinforcement Learning Approach for Dynamic Multi-objective Optimization.* Information Sciences 546: 815-834.

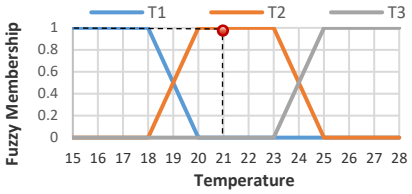
Appendix

1. <u>Fuzzification:</u> Given *case 1*'s input data: Temp_DataItem1 = 19.5 °C, and Temp_DataItem2 = 21 °C, we compute the corresponding fuzzy membership values following the *temperature* fuzzy functions in Figure 6.b (reported below):

- For Temp_DataItem1:
  $F_{T1}(19.5) = 0.25$, $f_{T2}(19.5) = 0.75$, and $f_{T3}(19.5) = 0$



- For Temp_DataItem2:
  $F_{T1}(21) = 0$, $f_{T2}(21) = 1$, and $f_{T3}(21) = 0$



2. <u>Condition-Action rules:</u> Based on the input membership values, the following condition-action rules are invoked:
- ***Rule 4***: T1(Temp_DataItem1) ∧ T1(Temp_DataItem2) ⇒ *Duplicate*(DedupStatus)
- ***Rule 5***: T2(Temp_DataItem1) ∧ T2(Temp_DataItem2) ⇒ *Duplicate*(DedupStatus)

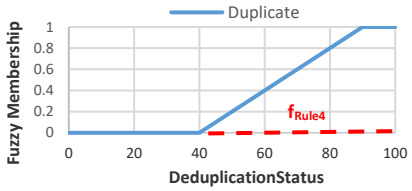
3. <u>Inference:</u> Applying *Mamdani*'s inference mechanism:

Rule 4:
Executing the AND fuzzy operator:

$$min\left(f(19.5)_{T1}^{Temp\_DataItem1}, f(21)_{T1}^{Temp\_DataItem2}\right) = min(0.25, 0) = 0$$

Executing the implication fuzzy operator:

$$f_{Rule4} = min\,(0, Duplicate(DedupStatus))$$

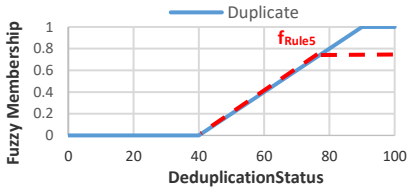


Rule 5:
Executing the AND fuzzy operator:

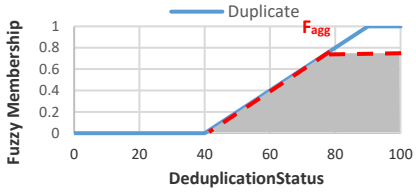
$$min\left(f(19.5)_{T2}^{Temp\_DataItem1}, f(21)_{T2}^{Temp\_DataItem2}\right) = min(0.75, 1) = 0.75$$

Executing the implication fuzzy operator:

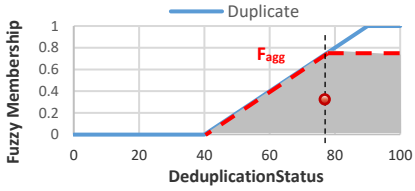
$$f_{Rule5} = min\,(0.75, Duplicate(DedupStatus))$$



4. <u>Aggregation</u> 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).



5. <u>Defuzzification</u>: 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) as the agent's output = 78.2%.



6. <u>Result:</u> 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% ≥ $dedup_{threshold}$, the agent's final output becomes: dedupStatus = *duplicates*

Figure 27. Fuzzy redundancy detection process for the *temperature* sample case considered in our running example (cf. Table 8).