Data Resilience System for Fog Computing

Franklin Magalhães Ribeiro Junior[a,b], Carlos Alberto Kamienski[a]

[a] Federal University of ABC (UFABC), SP, Brazil
[b] Federal Institute of Maranhão (IFMA), MA, Brazil

ARTICLE INFO

Keywords:
Data reduction
Fog computing
Internet of Things
Resilience
Trustworthiness

ABSTRACT

Fog computing improves IoT systems by analyzing and storing data locally at the network edge. However, it is challenging to design a fog-based IoT system data flow as data transmissions must be agile and resistant to network failures and disconnections. Data collected by sensors must persist in the fog, even during a long disconnection period. When the connection is available again, the fog needs to send the data immediately to the cloud. This paper proposes and evaluates the Fog-DaRe system for supporting data flow resilience between fog and cloud during network availability and unavailability situations. Fog-DaRe allows data persistence in the fog and uses different compression techniques to reduce the data volume. We evaluate 10 data flow configurations in experiments with 5,000 simulated sensors, computing hardware, and network metrics. The Fog-DaRe strategy yields different tradeoffs for scenarios with network unavailability, lossy compression techniques, and data encryption. For example, our results reveal a reduction of at least 77.7% for fog-to-cloud batch transfer time and 81.5% for fog storage usage when the network is unavailable. When the network between fog and cloud is available, delay increases by 10% due to data buffering in the fog, but storage requirement drops 73.6%. Lossy data filtering yields a reduction of 83.3% in batch transfer time and 50% in storage. Also, the compression of encrypted data increases storage usage and batch transfer time by 125%, compared to plain data.

1. Introduction

The Internet of Things (IoT) collects, transmits, stores, and analyzes data generated by physical devices connected to a network [1]. An IoT system uses sensors that monitor the environment context by collecting data and performs actions through actuators. IoT systems are usually based on robust centralized cloud computing infrastructures to store and process data and rely on distributed fog computing resources.

A fog-based IoT system can analyze and store data locally at the network edge, regardless of the availability of an Internet connection to the cloud [2]. As the fog is at the edge of an inherently distributed IoT system, it can process sensor data locally and provide a faster response to actuators to change system behavior accordingly. Actions taken by actuators may be, for example, turning on or off an irrigation system or increasing and decreasing the intensity of a street lighting system [3]. Nevertheless, as the fog depends on a centralized view of the system located at the cloud, a current challenge is dealing with disconnections or network failures for ensuring a resilient data flow from sensors up to the cloud [4].

Resilient solutions for a fog data flow must deal with data from thousands of sensors and consider memory, storage, and processing constraints of its computing nodes [5] [6]. In the event of a network disconnection between fog and cloud, the fog needs to store the data temporarily and transmit it to the cloud when the connection returns. The longer the disconnection period, the larger the data volume stored in the fog. Therefore, the fog must deal with large data volumes to avoid storage overflow and long network delays when the connection returns. We designed the *Trustworthiness for IoT Framework* (TW-IoT) [7] that considers various data resilience possibilities for a fog-based IoT system data flow. However, TW-IoT focuses on general concepts and does not propose and evaluate specific mechanisms.

This paper proposes the Fog-DaRe (Fog-based IoT Data Resilience) system to provide data filtering and persistence mechanisms preliminarily introduced by the TW-IoT framework. Fog-DaRe maintains the data flow continuity in the communication between fog and cloud even under network disconnections. Fog-DaRe guarantees fog computing resilience, which is a vital trustworthiness requirement for computer systems[8]. It also supports the data flow in the fog stage, offering data resilience and filtering mechanisms when the network is available or not. A disconnection-aware store and forward mechanism can guarantee data resilience. On the other hand, we can deploy multiple data filtering mechanisms in the Fog-DaRe system. Currently, we implemented six filtering techniques: a) run-length-encoding (RLE) [9, 10, 11]; b) Zstandard [11, 12]; c) Symbolic Aggregate approXimation (SAX) [13, 14, 15, 16]; d) Piecewise Aggregate Approximation (PAA) [13, 16]; e) standard deviation based on the Chauvenet outlier removal technique [17, 18], and; f) a naïve filter based on the arithmetic mean.

We conducted a performance analysis study of Fog-DaRe using a smart farming context [19], for 5,000 simulated sensors, with ten data flow configurations, using the six filtering techniques, as well as a baseline case with no filtering. We evaluate these configurations in two network availability scenarios, including experiments with stable and unstable network connections. For the latter, we generate an inter-

*Corresponding author:
E-mail addresses: franklin.ribeirojunior@ifma.edu.br (F.M. Ribeiro Junior); carlos.kamienski@ufabc.edu.br (C.A. Kamienski)
ORCID(s): 0000-0002-2988-7878 (F.M. Ribeiro); 0000-0002-3087-9234 (C.A. Kamienski)

ruption followed by a reconnection. In both scenarios, we measure the network delay and throughput and the usage of CPU, RAM, and storage. We also measure the batch transfer time after the connection returns for the scenario with a disconnection period.

Our results reveal that the Fog-DaRe has a low impact on CPU and RAM usage. For the connection availability scenario, data filtering techniques have advantages for storage usage, imposing 10% more delay than applying no filtering method in the fog. On the other hand, we observe substantial benefits for the batch transfer time for the disconnection scenario. In the worst case, the Fog-DaRe causes a reduction of 77.7% in batch transfer time and 81.5% in storage usage, but it reduces the batch transfer time by 99% and storage in the best case% usage by 98.8%. Our analysis unveils that each filtering configuration exposes particular advantages and disadvantages according to different metrics. For example, using lossy or lossless compression techniques directly impacts the data reduction and batch transfer time. Compressing the received encrypted packets from mist demands less CPU and RAM usage.

To the best of our knowledge, this is the first study to address fog resilience, focusing on resource constraints, transmission delay, and fog-to-cloud batch transmission time, evaluating distinct compression techniques combined with data persistence mechanisms, using thousands of sensors in a smart farm scenario. Therefore, the main contributions of this paper are (i) to propose a fog data persistency solution that combines filtering to reduce the massive data volume generated by IoT applications, (ii) to evaluate the impact of data filtering on fog-to-cloud batch transfer time after a disconnection event (iii) to clarify the tradeoffs between storage and transmission delay for different filtering mechanisms in a resilient fog system, and (iv) to discuss the tradeoffs related to fog filtering mechanisms and their implications to fog computational resources.

We organize the rest of the work as follows: Section 2 presents the background, Section 3 describes the Related Work, and Section 4 introduces the Fog-DaRe system. Section 5 explains our research methodology for the performance analysis, where Section 6 presents the results. Section 7 discusses the results and lessons learned, and finally, Section 8 draws some conclusions and lines for future work.

2. Background

This section presents the stages of a distributed IoT system, resilience in IoT, and data filtering.

2.1. Stages of a Distributed IoT System

We consider a fog-based IoT system data flow passing through a 4- stage distributed infrastructure comprised of thing, mist, fog, and cloud [20]. Each stage corresponds to a set of devices, equipment, and software components with similar characteristics and roles (Fig. 1).

The thing stage contains sensors and actuators that collect data and act in an environment [4]. The mist stage is directly connected to the thing stage, providing communication to sensors and actuators. Mist computing represents another stage of a distributed IoT system between sensors and actuators and the fog computing infrastructure [21, 22]. Although it can be considered another level of a hierarchical fog solution, the mist usually plays the role of a field-installed radio gateway to provide communication to the connected devices [20]. Mist nodes can also process and store data in a limited way [22]. The fog stage deals with massive data and can analyze and store data locally at the network edge, independently of an active Internet connection. However, given the high number of fog nodes, they have resource constraints absent in the cloud [4]. The cloud stage provides virtualized hardware resources with high computing capacity to store and analyze large data volumes [23].

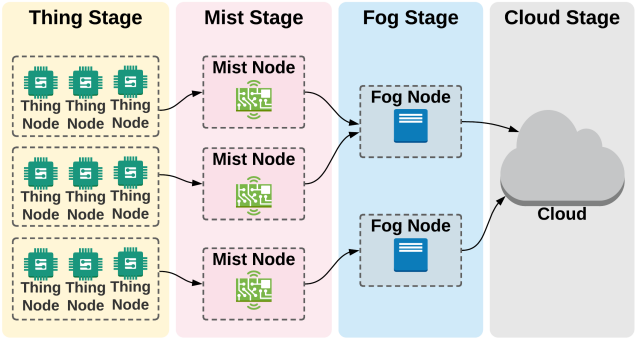

Figure 1: IoT Stages.

2.2. Data Flow Resilience

Resilience refers to the system's ability to prevent, mitigate, and resist failures [24]. Therefore, system resilience is part of the definition of trustworthiness [8], which deals with vulnerabilities. In an IoT system, resilience means supporting and recovering the communication, even with device malfunction or network interruptions [25]. However, enforcing resilience demands additional computing resources, and therefore an IoT system must be concerned about the typical resource constraints in the network edge, such as network bandwidth, battery, storage, RAM, and CPU) [26]. In other words, there is a trade-off between achieving higher resilience levels and using scarce computing resources for a non-functional requirement.

Under network disconnections, the fog-based IoT system can cope with this failure by maintaining service continuity at the data level. For example, the system can use a fog-based data persistence mechanism during a disconnection with the cloud. However, assuming that long disconnection periods may occur between IoT stages, the stored data can cause fog memory overflow and long data transmission delays. For that reason, the fog can use a data filtering mechanism to solve this problem. The same reasoning is valid for data disconnections between the mist and the fog.

2.3. Data Filtering

Data reduction (filtering, aggregation, and compression) mechanisms optimize and yield positive impacts in data flow between IoT stages [27, 28, 29, 30, 31, 32]. In an IoT healthcare system scenario, a fog node can use wavelet transforms to extract ECG features, reducing the data size [27, 28]. Data reduction mechanisms prevent the fog from transmitting a large amount of unnecessary or redundant data to the cloud [30]. The fog can use data filtering to aggregate data by categories, thus reducing the data size and data transmission rate [29, 32]. Additionally, compressing IoT data reduces the device's energy consumption because it reduces the transmission rate [28].

We refer to data reduction mechanisms (filtering, aggregation, and compression) as data filtering throughout this paper. Different techniques exist for data filtering, with differing performances and implications. Some techniques lose data (called lossy techniques), where other techniques (called lossless) preserve the whole dataset. For example, a naïve filtering method based on the arithmetic mean loses data because after computing this technique, the mean value represents the dataset, and the original data is lost. On the other hand, typical compression algorithms recover the entire dataset when uncompressed.

Computing the standard deviation can work as a lossy filtering technique, within a range c of standard deviations. The c constant delimits the records to be filtered, using the Chauvenet technique to remove outliers [17, 18]. Instead of eliminating outliers, it deletes the data outside the range c. Run-length-encoding (RLE) is a compression technique that aggregates data with repeated values [9, 10, 11]. We can also modify the RLE technique to classify and join the repeated data values in the same categories, preserving the data values' order in a data time series. The Symbolic Aggregate approXimation (SAX) technique has many variants, and it reduces the data size by aggregating similar data values in symbols, like a dictionary algorithm [13, 14, 15, 16]. Therefore, the SAX algorithm replaces numerical data, such as temperature and moisture, with a symbol. The Piecewise Aggregate Approximation (PAA) method minimizes the number of values in a data time series by aggregating values using a calculation, which results in a few values representing the original time series [13, 16]. Zstandard is a lossless compression algorithm to compress data files, and it keeps original data when uncompressing [11, 12]. The Zstandard compresses the data into a file, requiring decompression to access the original data file.

3. Related Work

There are studies on communication resilience for fog and edge computing systems [33, 34, 35, 36]. Some studies focus on replicating data from one fog node to others, which demands more computing resources from the IoT system as a whole [33, 34]. Jeong et al. consider self-healing mechanisms in case of fog failure by allocating a new fog node and using data redundancy or data migration [35]. Also, Al-Khafajiy et al. propose a load balancing mechanism between fog nodes to deal with a large amount of data received from end-node devices [36]. However, unlike our Fog-DaRe system, these investigations do not introduce mechanisms for detecting and recovering faults in the event of network disconnections between fog and cloud.

Delay (or Disruption) Tolerant Networks (DTN) can store data by a particular time during disconnection and send them through the network when the link is available [37]. Some studies relate DTN to fog-based IoT systems [38, 39, 40], but they only deal with delay or packet loss metrics, neglect fog resource constraints, and only cover particular scenarios. Luzuriaga et al. propose IBR-DTN, an architecture for MQTT with DTN to evaluate the communication between a Raspberry Pi gateway and sensor boards [38]. They vary the communication channel error by 0%, 25%, 50%, and 75% in three disconnection scenarios, no disconnection, 6-minute disconnection, and 12-minute disconnection. However, they only consider seven sensor boards in the experiment and mention that IBR-DTN has no persistent storage, using RAM for storing data.

Some studies use DTN in smart farming scenarios with mobile nodes, where nodes transmit data, whenever they encounter each other [39, 40]. Kulatunga et al. evaluate a real scenario by monitoring cows, where each cow carries data (generating 100kB every 5 minutes) and transfer the data to the farmer wearables or a gateway station. In this study, the authors verify the relation between the distance, network delay, and devices' energy consumption [39]. Castellano et al. evaluate an architecture using DTN with mobile nodes, mentioning that the fog can use filtering and aggregation for data cleaning, even though they do not apply these mechanisms [40]. They use simulation to evaluate the delivery time and delivery rate of bundles (packets), varying the connection probability between devices in 10%, 25%, and 40%. They also evaluate the same metrics during 1, 2, 4, and 6 seconds of connection time. They realized that this architecture replicates packets, which causes a higher usage of RAM.

Moura and Hutchison mention that edge network devices can *"aggregate and synthesize useful information from the received raw data"* to reduce the high data volume in IoT [41]. Some evaluate compression techniques to reduce the data in the IoT systems studies [9, 11, 12, 42, 43], even though they do not assess a scenario with the disconnection between the IoT stages. Spiegel et al. propose the RLBE algorithm, a modification in the RLE technique, to an IoT scenario, comparing RLBE against other lossless compression algorithms and identifying that RLBE saves 60% of the energy required the data to arrive in the end-nodes (thing stage) [9]. However, the research does not consider any resilience impact.

Routray et al. conceptually compare different compression algorithms in IoT [11]. They mention that it is a challenge to improve the network performance in low bandwidth scenarios. A way to aim this goal is by reducing the irrelevant information transmitted. They mention the Zstandard, LZ4, and LZO as algorithms which demand low bandwidth, and they report the edge computing should compress the data

because the thing nodes have no computing power to deal with a large amount of data.

Gia et al. evaluate lossless compression algorithms (LZ4, LZW, Huffman, and Zstandard) in edge/fog computing with different devices (Raspberry Pi 3B, Intel UP, Intel i5, and UP gateway) [12]. The authors perceived that the Zstandard algorithm shown a better compression rate. However, they only evaluate the compression rate and compression speed, not the compression impact for IoT resilience.

Chandak et al. and Blalock et al. propose algorithms to compress multivariate data time series, using prediction models to reconstruct the data (based on previous data samples) [42, 43]. They use datasets from IoT scenarios to evaluate their algorithms against other compression techniques, obtaining a compression and decompression ratio and speed. Besides, neither Chandak et al. nor Blalock et al. proposed the data compression for a resilient IoT data flow.

Fu et al. [44] proposed F2MC, a solution for managing data between fog and multi-clouds that deals with data fault tolerance and data reduction. However, their solution only uses a unique data reduction mechanism, and the authors do not evaluate F2NC in a scalable scenario with thousands of sensors sending data in real-time nor consider network device constraints like LPWAN. Sinaeepourfard et al. [45] present a data management framework (F2C) to collect and reduce data received by thousands of sensors. Their study shows that the data volume can be reduced in IoT systems by eliminating redundant data with a well-known ZIP compression algorithm. However, they do not consider other aspects of resilience in their evaluation, including the impact of different data reduction methods in transmission speed after a cloud disconnection and fog computational resources.

4. The Fog-DaRe System

This section presents the Fog-DaRe system that focuses on data communication resilience and fog constraints to improve the communication between fog and cloud, as a proof of concept for the data persistence and filtering of the TW-IoT framework [7].

4.1. Fog-DaRe Requirements

The Fog-DaRe system specifies and implements data persistence and filtering mechanisms of TW-IoT [7], thus satisfying data recoverability requirements and reducing data volumes transmitted throughout the IoT stages. The system implements the following requirements:

(i) Recoverability: Fog-DaRe guarantees recoverability (survivability) by avoiding data loss between fog and cloud, even under disconnections. In the event of a connection failure between fog and cloud, the fog stores data in a queue and continually checks the connection's availability. When the link returns, the fog sends the stored data to the cloud.

(ii) Data filtering: Fog-DaRe reduces the data volume using filtering techniques in the fog. For this reason, it reduces the time to send to the cloud all data stored in the fog after a reconnection event.

(iii) Resource constraints: Fog-DaRe uses as few computing resources as possible because our system does not use additional computing resources to replicate or migrate data or fog nodes.

4.2. Fog-DaRe Data Flow

Fog-DaRe guarantees a reliable data flow in a multi-stage IoT system by ensuring data flow continuity by providing persistent data storage in the fog. To this end, two mechanisms are available: a) data filtering for reducing the data volume transmitted through the IoT system stages and: b) data persistence for storing data locally in the fog during network disconnections (Fig. 2).

The data flow passes through the four IoT stages, namely, (i) thing nodes, (ii) mist nodes, (iii) fog nodes, and (iv) cloud. The Fog-DaRe data path works as follows: (i) thing nodes collect and send data to the mist; (ii) mist nodes forward the data to the fog; (iii) fog nodes receive the data via different protocols, such as MQTT or LoRaWAN, (iv) fog workers store the original data in the *Raw Data Queue*, even under a cloud network disconnection event, (v) fog worker filter the data and insert it in the *Filtered Data Queue* and if the cloud connection is available, (vi) fog nodes send the filtered data to the cloud (Fig. 2).

A fog node can have multiple fog workers, and one fog worker exclusively stores and filters data from one mist node. Besides, each fog worker has different data flow configurations. Individually, each fog worker can implement none, one, or two techniques for data filtering. As shown in Fig. 2, fog worker 1 receives data directly from mist node 1, then stores, filters, and sends the data to the cloud. For example, fog worker 2 receives data from the respective mist nodes via a LoRaWAN server [46].

The benefit of data filtering is to reduce the packet transmission delay after returning the fog connection. When the link returns after a long disconnection period, a fog worker without any data filtering techniques takes longer to send all data to the cloud. Delays in data transmission can cause delays in IoT system decisions, and consequently, the IoT system makes erroneous decisions. Therefore, transmission speed impacts the system's trustworthiness.

However, filtering data can also pose some difficulties to specific IoT applications when the total data volume is not preserved. For example, healthcare applications usually need the original data not to cause erroneous decisions. Therefore, a filtering method should not use lossy compression algorithms (as SAX or PAA) as they do not keep the original data [29]. One should only use lossless compression algorithms to recover the original data (like Zstandard) whenever needed by the application.

4.3. Fog-DaRe Data Filtering Methods

The Fog-DaRe currently provides six data filtering techniques (Section 2.3), the RLE, Zstandard, SAX, PAA, filtering by calculating the standard deviation (based on the Chauvenet outlier removal technique) and filtering by calculating the arithmetic mean. Our system starts the data filtering upon reaching the queue threshold, i.e., when the Raw

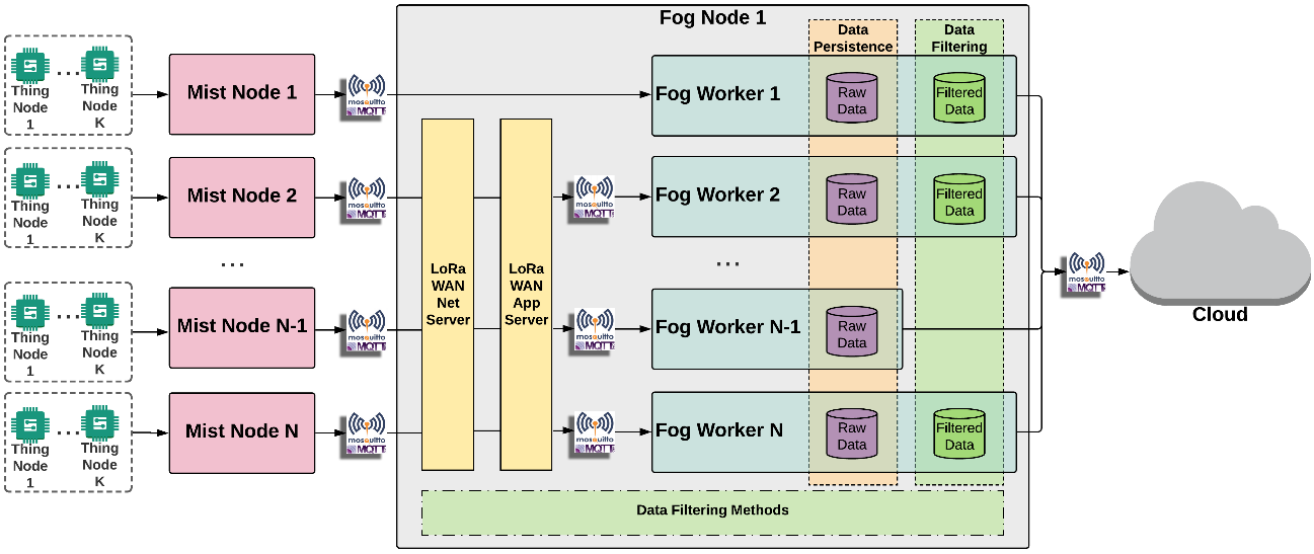

Figure 2: Fog-DaRe Data Flow.

Data queue has a minimum number of packets for starting the filtering process. We use the same queue threshold for every filtering method, except for Zstandard during a disconnection event, which compresses the whole data volume.

We modified RLE to classify numeric data as categories, where the first digits of the data values present each category. For example, if the value is 14.96423, we consider that the filter uses five digits (14.96) as a category to classify the data using the RLE algorithm. Our modified RLE only stores the category digits (14.96) and their number of occurrences in the dataset instead of keeping the whole number. Therefore, in our experiment, we treat RLE as a lossy compression algorithm.

4.4. Fog Worker Implementation

We implemented our fog workers using two threads (Fig. 3), where the first thread runs the *Accumulator Component* and executes the Mosquitto MQTT broker [47] to receive data from the mist. On receiving data from the mist, a fog worker stores it in the *Raw Data Queue*, using the *pqueue* library [48]. The second thread corresponds to the *Flow Component*, which uses the data filtering methods and checks the connection status to send the stored data to the cloud.

The *Flow Component* contains the modules named *Connection Verifier*, *Data Filterer*, and the *Data Sender*. The *Connection Verifier* uses the RTT calculation to check the availability of the connection to the cloud frequently. In a network disconnection, the *Accumulator Component* continues to store the data in a queue persistently. Then, when the link returns, the cloud receives the data from the fog worker through the *Data Sender*. The *Data Filterer* is responsible for filtering the stored data, and it is indifferent to a disconnection event. There are four operation modes for data filtering:

- Single Connected Filtering: a single filtering technique under regular system conditions, represented by the *filtering method A* (Fig. 3);
- Single Disconnected Filtering: a single filtering technique only when the fog disconnects from the cloud, represented by the *filtering method B* (Fig. 3);
- Double filtering: a filtering technique used for regular operation and another for network disconnection (using both *filtering methods, A and B*);
- No filtering: our baseline option.

Assuming that the connection with the cloud communication is available, the fog worker checks whether it filters the data or not (Fig. 3). Assuming a *No filtering configuration*, the fog worker reads the *Raw Data Queue* and sends data to the cloud via the *Data Sender* (Fig. 3). When the data flow configuration uses data filtering, the fog worker waits for the queue threshold to perform the filtering (Section 4.3). If so, it reads the *Raw Data Queue*, filters the data using one of the filtering techniques (by *filtering method A*), and finally stores it in the *Filtered Data Queue*. After that, the fog worker searches for filtered data stored in the *Filtered Data Queue* and sends it to the cloud via the *Data Sender*.

For the scenario where fog and cloud are disconnected, the fog worker may filter the data or not (Fig. 3). For a *No Filtering configuration*, the fog worker waits for the connection to return. For a configuration with data filtering and when the *Raw Data Queue* reaches the queue threshold (Section 4.3), the fog worker reads the data from the *Raw Data Queue*, filters it (using *filtering method B*), and stores it in the *Filtered Data Queue*. Thus, when the connection returns, the fog worker sends the filtered data to the cloud by the *Data Sender*.

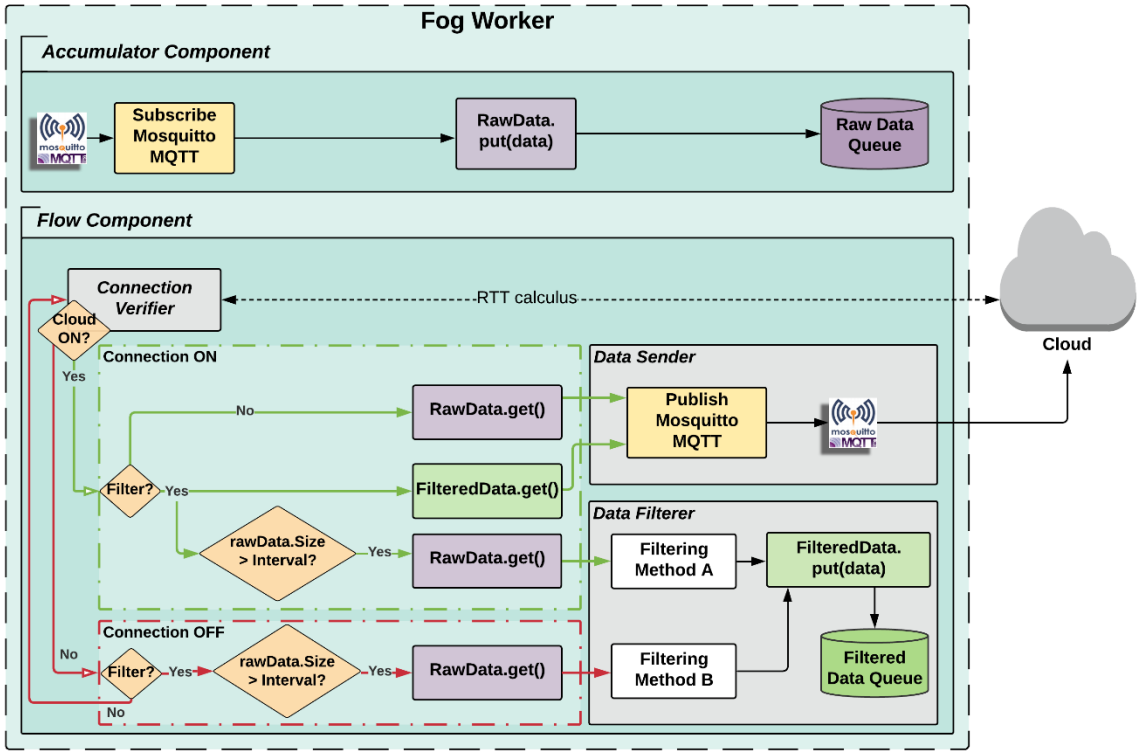

Figure 3: Fog Worker Data Flow.

5. Research Methodology

Our experimental environment emulates a fog-based IoT system that duly represents a smart farming system [19]. Fig. 4 depicts our testbed with the testing environment composed of four IoT stages: (i) Thing: the SenSE sensor simulator [49] generates workload representing LoRaWAN sensors that capture temperature and humidity data; (ii) Mist: SenSE also abstracts the LoRaWAN gateway in the mist, since our focus is to analyze the connection between fog and cloud; (iii) Fog: a fog worker receives data from the mist using the ChirpStack LoRaWAN Server [46]; (iv) Cloud: runs a Mosquitto MQTT broker and a FIWARE IoT Agent [50] representing a data consumer.

The testbed comprises two physical servers running three virtual machines for playing the role of thing/mist, fog, and cloud nodes. The thing/mist and fog nodes run on a Linux Ubuntu 18.04 operating system, with an Intel i5-8265U processor at 1.60GHz and 8GB RAM. The cloud VM runs on a Linux Ubuntu 18.04 operating system, with four 2.4GHz CPU cores and 8GB RAM. Both physical servers are connected by a network, with an average RTT of 5.67ms $\pm$ 3.3ms.

The SenSE simulator generates encrypted LoRaWAN packets and publishes them via MQTT in the fog, where the ChirpStack server receives them. ChirpStack decrypts the packet payload and passes it to the fog worker to be stored, filtered, and transmitted to the cloud.

5.1. Metrics

During the experiments, we collect and compute the following metrics:

- Delay: The end-to-end delay (one-way delay) measured between the thing and the cloud states. We record timestamps at packet creation for each LoRa packet generated in the thing stage by SenSE.
- CPU and RAM Usage: CPU and RAM usage is measured every second for Mosquitto, ChirpStack, and Fog-DaRe processes using the *ps* Linux command. As the CPU has eight cores, we convert the values accordingly.
- Packet loss rate: computed by comparing the number of packets received by the Mosquitto brokers installed in the fog and the cloud.
- Storage Usage: the disk or SSD space used in the fog for data storage.
- Throughput: This metric is the data throughput received in the cloud and measured every second by the *ifstat* Linux command.
- Batch transfer time: time for all packets stored by Fog-DaRe in the fog to arrive in the cloud after the network connection returns. It is the subtraction of the timestamps of the last and the first packets to arrive in the cloud.

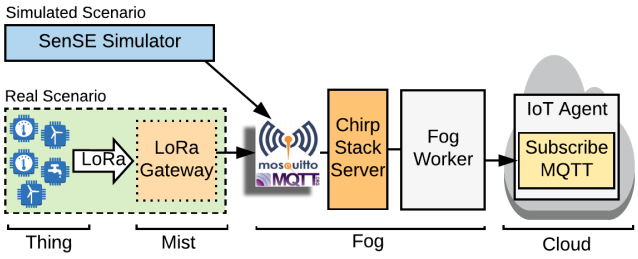

Figure 4: Data Flow for the Experimental Environment.

5.2. Input Dataset and Workload Generation

We used a dataset containing air humidity and temperature values from Spain [51] as the basis for our synthetic input dataset. This dataset is not compatible with a scenario with 5,000 sensors because it includes records measured every minute. Using the original data as a parameter (one record per minute), we generated 5,000 datapoints every ten minutes, following a fitted Gaussian distribution. In other words, we simulated 5,000 sensors sensing the environment every ten minutes and sending data to the cloud via the Fog-DaRe system. We also created a second dataset with identical humidity and temperature values to evaluate the system performance under typical circumstances of an agricultural setting [32].

The SenSE simulator reads the input data and generates the experiments' workload simulating sensors in the thing stage, each sensor sending data every ten minutes. Each record is encapsulated in a LoRaWAN packet with approximately 481 bytes.

5.3. Experimental Configurations

We evaluate 10 data-flow configurations for the Fog-DaRe features of resilience and filtering. Resilience is tested by disconnecting the network between fog and cloud for a period and reconnecting it again. We explore two scenarios of network connection: a) Scenario 1 (Network-On), where the network connection between fog and cloud is continually operating throughout the 60 seconds of the experiment; b) Scenario 2 (Network-On-Off-On), where the network between fog and cloud starts connected for 15 seconds, faces a period of disconnection (we tested 5 and 30 minutes) and reconnects again for 5 minutes.

In some cases, several sensors report identical values for extended periods, such as soil moisture, air humidity, temperature, or even binary values like a light detector (on or off). For that reason, we considered in our experiments two configurations that receive identical values of temperature and air humidity as input data.

For each network connection scenario, we explore ten cases of filtering techniques, with a queue threshold of 25 packets:

1. *NoFilter*: the fog worker does no use any filter (Fig. 5);
2. *Mean*: uses the arithmetic mean as the filtering method (Fig. 6);
3. *StdDev*: uses the standard deviation method based on the Chauvenet technique for data filtering. This technique has a constant c for denoting a range of standard deviations, which here is c=1 (section 2.3);
4. *Zstd*: uses the Zstandard compression algorithm for data filtering (Fig. 6);
5. *ZstdEncrypt*: uses the Zstandard compression algorithm for data filtering but does not use the LoRaWAN ChirpStack server as they use encryption in the thing stage. In this configuration, the fog worker compresses the original encrypted data that arrive in the fog (Fig. 7);
6. *ZstdIdentical*: uses the Zstandard compression algorithm for data filtering operating on the dataset with identical values;
7. *RLE*: uses the run-length-encoding (RLE) algorithm for data filtering
8. *RLEIdentical*: uses the RLE algorithm for data filtering operating on the dataset with identical values;
9. *SAX*: uses the SAX algorithm for data filtering;
10. *PAA*: uses the PAA technique for data filtering;

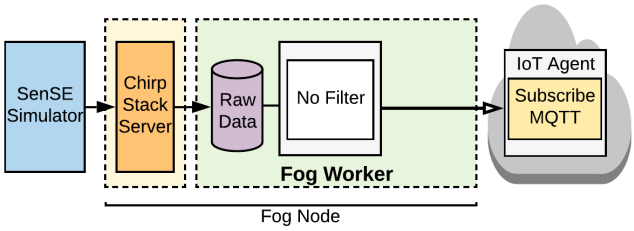

Figure 5: Data Flow - *NoFilter* & ChirpStack LoRaWAN Network Server.

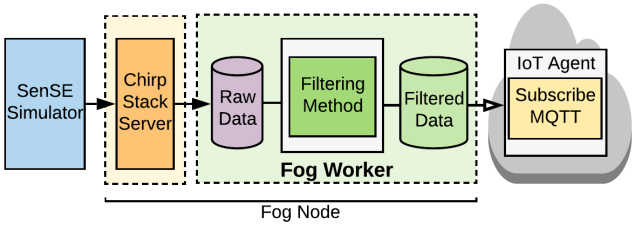

Figure 6: Data Flow – Filtering Methods & ChirpStack LoRaWAN Network Server.

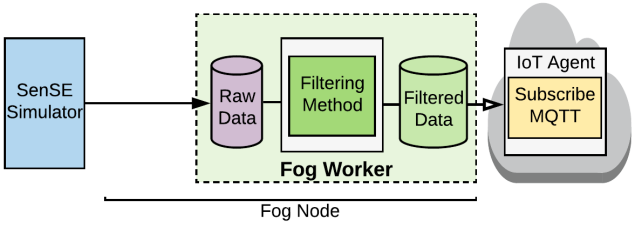

Figure 7: Data Flow - *ZstdEncrypt* Configuration.

6. Results

Our evaluation considers 30 replications of each experiment, from which we calculate the arithmetic mean of the

Table 1
Network Delay, Throughput, CPU, RAM, and Storage Usage for Scenario 1.

Configuration	Delay (seconds)	Throughput (kB/s)	CPU Usage (%)	RAM Usage (%)	Storage Usage (kB)
RLEIdentical	3.56 ± 0.05	2.71 ± 0.02	5.78 ± 0.10	1.12 ± 0.01	1.28 ± 0.01
Mean	3.58 ± 0.06	2.50 ± 0.09	5.79 ± 0.10	1.13 ± 0.01	1.34 ± 0.01
RLE	3.62 ± 0.05	2.47 ± 0.10	5.71 ± 0.11	1.12 ± 0.11	4.01 ± 0.04
Zstd	3.63 ± 0.04	2.69 ± 0.06	5.67 ± 0.11	0.93 ± 0.01	8.77 ± 0.09
ZstdIdentical	3.62 ± 0.05	2.78 ± 0.07	5.77 ± 0.10	0.93 ± 0.01	2.68 ± 0.03
ZstdEncrypt	3.59 ± 0.04	3.12 ± 0.05	1.27 ± 0.01	0.24 ± 0.001	29.15 ± 0.16
SAX	3.59 ± 0.04	2.50 ± 0.05	6.65 ± 0.11	2.73 ± 0.14	4.97 ± 0.05
PAA	3.63 ± 0.05	2.37 ± 0.06	7.67 ± 0.24	2.63 ± 0.01	6.95 ± 0.07
StdDev	3.62 ± 0.05	2.90 ± 0.07	5.84 ± 0.11	1.13 ± 0.01	13.29 ± 0.46
NoFilter (1 packet)	0.30 ± 0.01	7.89 ± 1.16	5.71 ± 0.12	0.93 ± 0.01	110.50 ± 0.18
NoFilter (25 packets)	3.24 ± 0.07	7.89 ± 1.16	5.71 ± 0.12	0.93 ± 0.01	110.50 ± 0.18

metrics, also calculating the mean and standard deviation of these 30 means. Having these statistics, we calculate the 95% asymptotic confidence intervals. We do not present results for packet loss as the loss rate was 0% in all experiments.

6.1. Scenario 1: Network Availability between Fog and Cloud

Scenario 1 considers a stable network connection during the experiments, and the worker filters the data every 25 packets sending the resulting filtered data immediately to the cloud. As introduced in section 5.3, we used nine filtering methods: *Mean, StdDev, Zstd, ZstdEncrypt, ZstdIdentical, RLE RLEIdentical, SAX,* and *PAA*. Besides, we also conducted experiments with *NoFilter* as the baseline. In this scenario, we also calculate the delay for every 25 packets using *NoFilter* to compare it to the other configurations.

Table 1 shows the packet delay metric, where we can observe that all nine filtering techniques render similar values with overlapping confidence intervals. On the other hand, when the fog forwards packets to the cloud with *NoFilter* (even considering 25 packets), the delays cost 10% less than the filtering techniques. One can realize that four seconds may be considered an acceptable delay for most applications, but not for all of them. For example, connected cars need a real-time response from the computing infrastructure, which may require fast fog-based processing times.

The *NoFilter* configuration presents, on average, the highest value for throughput compared to the other configurations. The nine data filtering configurations revealed similar results for throughput, from which *Zstd* and its variants *ZstdIdentical* and *ZstdEncrypt* achieved the highest throughput. The reduction factor achieved by the filtering techniques varies from 2.53 (7.89 / 3.12) for *ZstdEncrypt* to 3.33 (7.89 / 2.37) for *PAA* (Table 1). Even though the overall bandwidth required is low, one must consider that our results represent an agricultural setting where sensors send data every 10 minutes. In a vast IoT deployment and more constant data transmission, reducing the network usage two to three times may significantly differ.

The *SAX* and *PAA* configurations recorded the highest CPU usage, an average of 6.65% ± 0.11% and 7.67% ± 0.24%, respectively (Table 1). The *ZstdEncrypt* configuration demands less CPU usage than the others because it is the only configuration that does not use a LoRaWAN Server on fog node (Section 5.3), in contrast to the others that need the LoRaWAN Server to decrypt the payload. The remaining configurations present a statistical tie for CPU usage. We perceive the *Zstd* demands at least four times higher CPU than *ZstdEncrypt* (5.67/1.27), and it happens because *Zstd* additionally uses the ChirpStack LoRaWAN Server.

The *SAX* and *PAA* configurations had the highest average percentage values for RAM usage. The *ZstdEncrypt* configuration registered the lowest RAM usage (Table 1) by compressing the encrypted data. Contrasting *Zstd, ZstdIdentical*, and *NoFilter* with *ZstdEncrypt*, we evidence that *ZstdEncrypt* achieves a 3.87 (0.93/0.24) reduction factor for RAM usage compared to the other configurations.

We observe that when the input data have identical temperature and humidity values, the configurations that use the Zstandard and RLE algorithms require less storage usage than when the input data have different values. We observe that even the filtering techniques with the highest storage usage like *ZstdEncrypt, StdDev*, or *Zstd* have a reduction in storage usage by 3.79 (110.5/29.15), 8.31 (110.5/13.29), and 12.59 (110.5/8.77), respectively. Therefore, even when filtering data with a lossless compression algorithm, FogDaRe reduces the storage needs for our evaluated input data at least twelve times.

6.2. Scenario 2: Network Unavailability between Fog and Cloud

We conducted two experiments for the disconnection scenario between fog and cloud, each with distinct periods of five and 30 seconds. In both cases, the connection returns after the disconnection period. Fig. 8 shows that the batch transfer time has a proportional behavior for the 5-minute and 30-minute disconnection experiments. We also perceive a similar behavior for storage usage in both experiments (Fig. 9) and scenario 1 (Table 1).

When the connection returns, the *NoFilter* configuration has the highest batch transfer time (Fig. 8). For the configu-

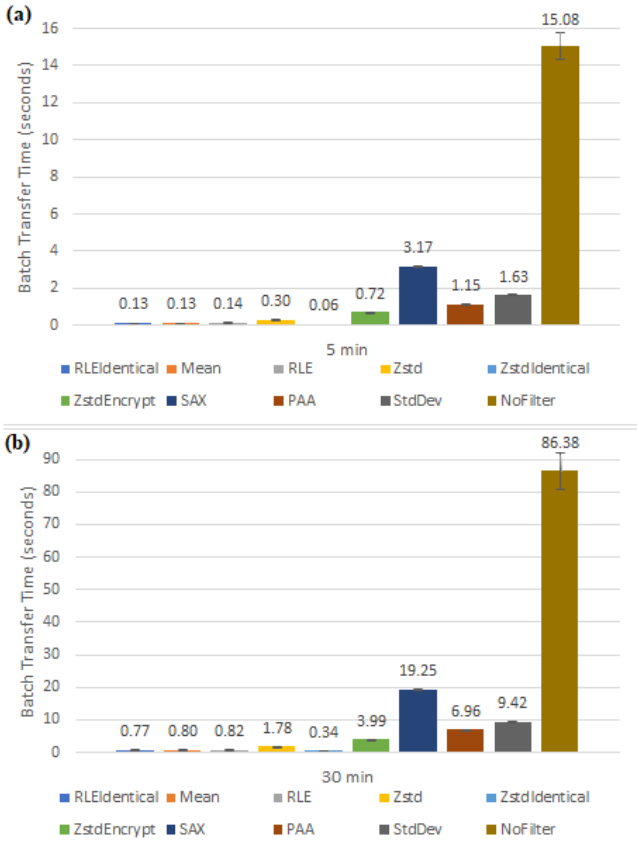

Figure 8: Batch transfer time after the connection return, with (a) 5 minutes disconnection and with (b) 30 minutes disconnection (different scales).

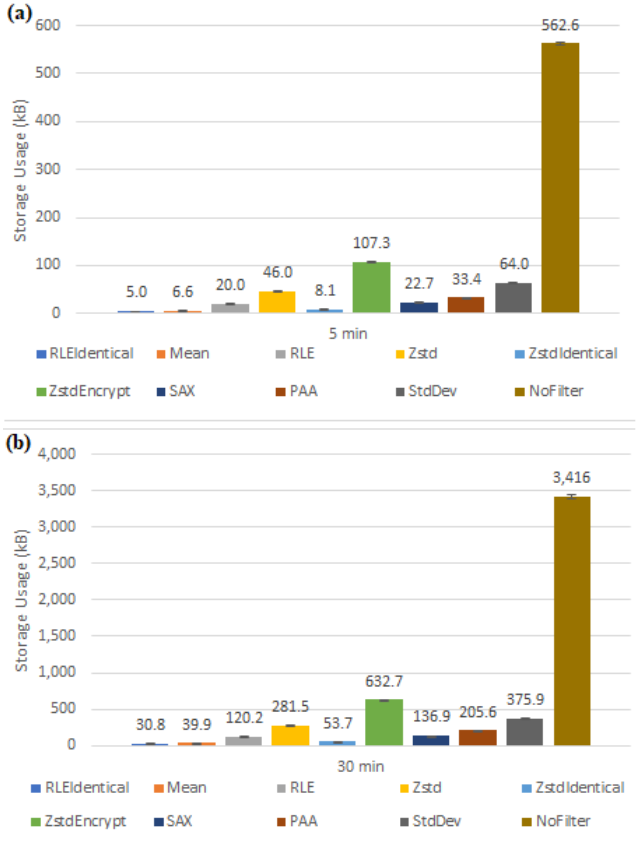

Figure 9: Storage Usage in Scenario 2, with (a) 5 minutes disconnection and (b) with 30 minutes disconnection (different scales).

rations with different input data, *Mean* and *RLE* present the lowest batch transfer times. Remarkably, the values achieved 116 (15.08/0.13) and 107.97 (86.38/0.8) times lower than using *NoFilter* for 5 and 30 minutes disconnection, respectively. However, *ZstdIdentical* achieved the lowest batch transfer time for identical input data (Fig. 8a).

For scenario 1 (Section 6.1), *NoFilter* presents the lowest delay, though requiring higher storage space. However, during disconnection, *NoFilter* storage usage is cumulative, increasing faster than the other configurations with time. The consequence is more data to be transmitted to the cloud when the connection returns. For this reason, when the link returns, the all-other filtering configurations present a lower batch transfer time than *NoFilter* in scenario 2.

RLEIdentical requires less storage space for both disconnection experiments, 5 and 30 minutes (Fig. 9). However, the *Mean* configuration demands less storage usage for both disconnection situations with different input data, with 6.6kB $\pm$ 0,01 for the 5 minutes (Fig. 9a) and 39.9kB $\pm$ 0,09 for 30 minutes (Fig. 9b). We observe data reductions for all filtering configurations comparing to *NoFilter*. The storage usage reduction is more evident when we compare *StdDev* or *ZstdEncrypt* to *NoFilter*, where we perceive a data reduction of 5.39 (3.416/632.7) and 9.08 (3.416/375.9) for the 30-minute disconnection experiment.

ZstdEncrypt achieves the lowest CPU (Fig. 10) and RAM usage (Fig. 11) for disconnection times. In contrast, *PAA* and *SAX* had the highest CPU and RAM usage. The *NoFilter, RLEIdentical, RLE, Mean, Zstd, ZstdIdentical*, and *StdDev* configurations had a statistical tie for CPU with 30 minutes disconnection (Fig. 11b). *NoFilter* achieved the lowest RAM usage for 30 minutes (Fig. 11b) among the LoRaWAN Server configurations. The exception is *ZstdEncrypt* that does not use a LoRaWAN Server. Therefore, even in the worst case, Fog-DaRe filtering considerably reduces data storage usage and batch transfer time.

6.3. Estimated Behavior for Long-Time Disconnections

In sections 6.1 and 6.2, we observed that the storage usage and batch transfer time metrics increase linearly with time. As the disconnection time rises, the fog may run out of storage capacity, and the time for sending data after a reconnection may create bottlenecks in the network between cloud and fog. Also, it can impact the update of analytical or data-oriented application models, rendering outdated models. Thus, we estimated the behavior for these metrics varying the disconnection time between 5 minutes and 24 hours. In a smart farming scenario, the connection may take some hours to return after a disconnection. Also, we be-

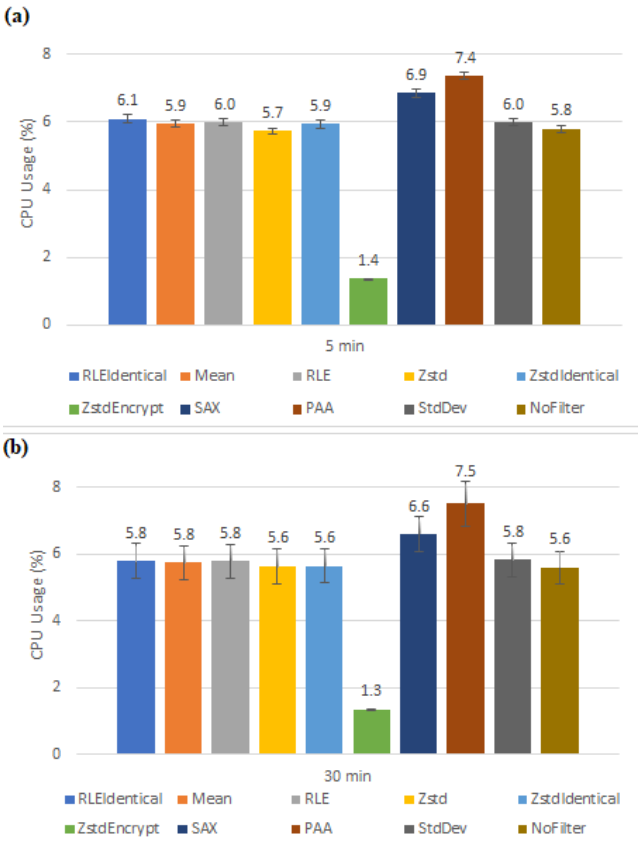

Figure 10: CPU Usage in Scenario 2, with (a) 5 minutes disconnection and (b) with 30 minutes disconnection.

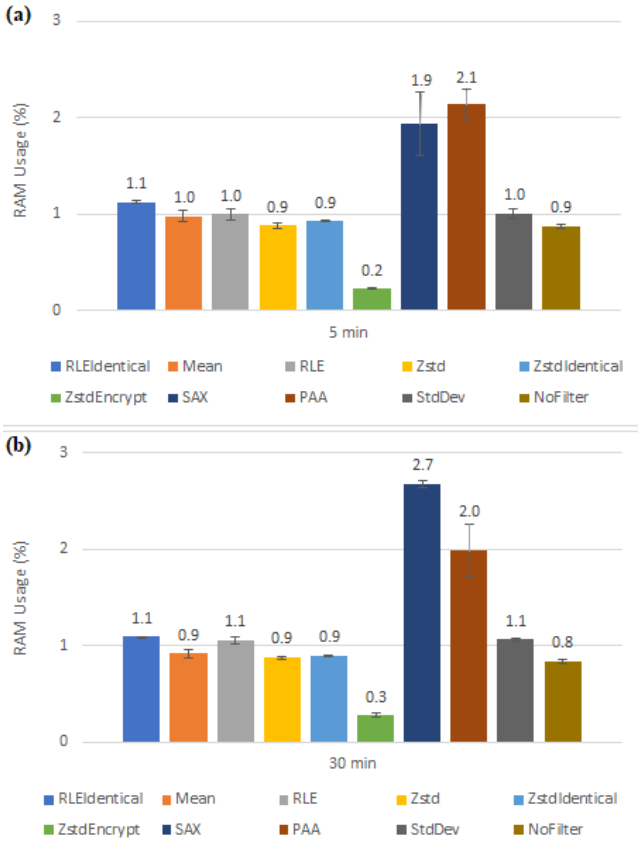

Figure 11: RAM Usage in Scenario 2, with (a) 5 minutes disconnection and (b) with 30 minutes disconnection.

lieve 24-hour disconnection is reasonable because, in other IoT applications, the fog can process the local data to make decisions and keep the system working for a period without cloud communication. However, the disconnection time is relative, and it depends on the fog constraints.

Fig. 12 depicts that *NoFilter* may impose a batch transfer time of 4,146.3 seconds for a 24-hour disconnection. It means that the fog needs approximately 69 minutes for sending all stored data to the cloud after the connection returns. Even *Zstd*, which preserves the original data after decompression and has not the lowest batch transfer time, takes 85.2 seconds to transfer the data (Fig. 12), a reduction factor of 48.77 (4,146.3/85.2).

The *Mean* configuration has the lower batch transfer time after a 24-hour disconnection, with varied input data. However, when the input data is identical, *ZstdIdentical* achieved the lowest time with 16 seconds. We also perceive that even compared with the worst case, using *SAX*, *NoFilter* takes 4.48 (4,146.3/923.9) times higher to transfer the data to the cloud (Fig. 12).

NoFilter required 163.97 MB of storage during a 24-hour disconnection (Fig. 13), but even *Zstd* used only 8.2% (13.49/163.97) of the necessary space by *NoFilter* (Fig. 13). Even for the worst cases, using *ZstdEncrypt* or *StdDev, NoFilter* uses 5.39 (163.97/30.38) and 9.08 (163.97/18.04) times the storage space. Therefore, the configurations with filtering techniques significantly impact storage usage comparing to the *NoFilter* configuration.

For datasets with identical input data, *RLEIdentical* has the lowest storage usage (1,478 MB), which is 3.87 (5,731 /1,478) times lower than *RLE* with different input data values. This behavior happens because the RLE algorithm combines the amount of identical data into a single category. The *ZstdIdentical* needs only 19% (2,575/13,488) of the storage used by *Zstd*. However, when the input data has different values, the *Mean* configuration had the lowest storage usage (1.916 MB).

7. Discussion

After analyzing the results, we observe no significant advantages for CPU and RAM usage between configurations, except for *ZstdEncrypt* that does not use ChirpStack. For this reason, we remark that the ChirpStack LoRaWAN server considerably increases CPU and RAM usage in the fog node.

We also realized that most filtering configurations present lower storage usage and lower batch transfer times than *ZstdEncrypt*. Still, they need to decrypt data in the fog, making it susceptible to intruders. In contrast, *ZstdEncrypt* keeps data confidentiality and does not risk the data payload because it compresses the encrypted data received from the mist. Therefore, there is a trade-off between confidentiality

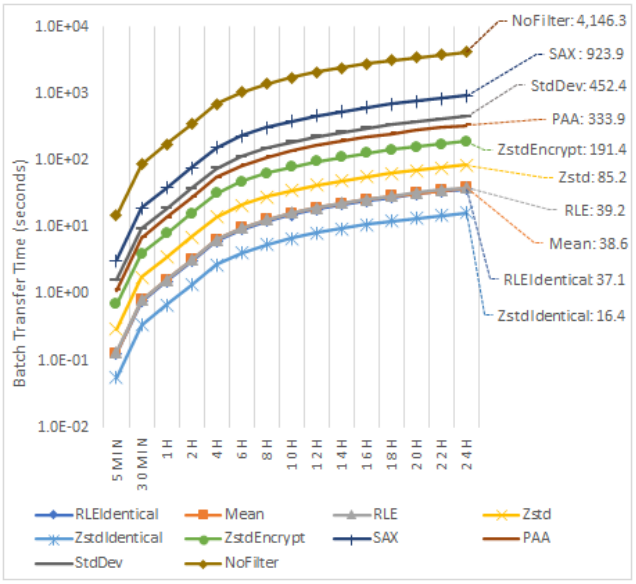

Figure 12: Projection of batch transfer time after the connection return, with 24-hour disconnection (logarithmic scale).

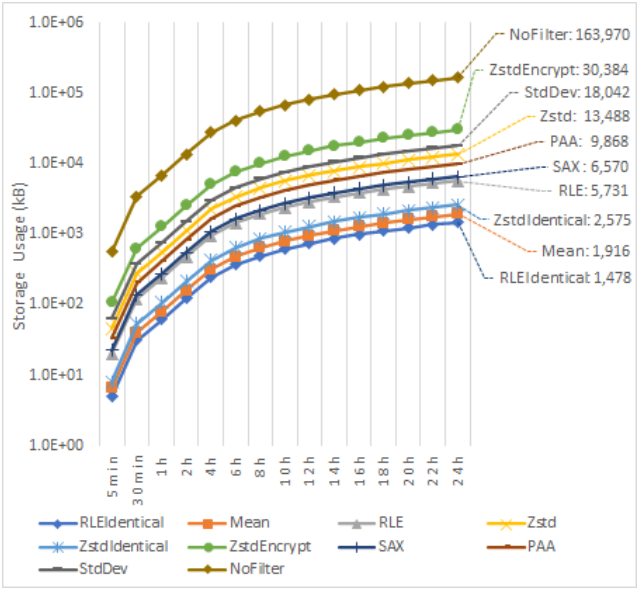

Figure 13: Projection for Storage Usage with 24-hour disconnection (logarithmic scale).

and storage usage, and batch transfer time.

For the scenario where the network is continuously available (section 6.1), we observe that the data filtering techniques reveal advantages for the storage usage compared to *NoFilter*. However, suppose the connection is continuously available. In that case, *NoFilter* always introduces the lowest delay because the fog only filters data after when the queue threshold is reached, which negatively impacts the delay of filtering techniques. Therefore, there is a trade-off between packet delay and storage usage.

The batch transfer time and storage usage metrics have significant variations between different configurations in the disconnection scenario, where it is advantageous to use data filtering compared *NoFilter*. Filtering techniques achieve lower batch transfer times and storage usage compared to *NoFilter*. This advantage is particularly evident when the disconnection time increases because the longer the fog remains disconnected, the more considerable the amount of data to be transferred to the cloud in the future.

In the disconnection scenario, the configurations with lossy data filtering had the lowest storage usage, confirming the intuition. However, this type of filtering suppresses details as it does not keep the original data, which may cause wrong decisions by the IoT system. The *Zstd* configuration uses the Zstandard lossless compression algorithm, but its storage usage and batch transfer times are higher than other configurations that use lossy filtering techniques. The noticeable exception is *ZstdIdentical*, but it loses generality as identical data requirements limit the application scope. Therefore, there is a trade-off between lossless and lossy filtering in the fog because lossy compression techniques need lower storage and transfer batch data faster to the cloud. Still, it does not preserve the original dataset.

Also, the configurations with the lowest storage usage do not have the lowest batch transfer times. Therefore, there is a trade-off between the different Fog-DaRe configurations. The trade-off also exists for the metrics evaluated and for the data details in each filtering technique. Consequently, developers need to analyze the application domain to determine which data flow configuration best suits each IoT system.

Finally, comparing Fog-DaRe to other solutions, we observed that data reduction rates vary between 81.5% and 98.8%, whereas F2MC [44] achieved 75% and F2C [45] 78%. This difference is because Fog-DaRe uses more than only one data reduction method. Also, the Fog-DaRe data filtering mechanism is configurable, and some filtering methods compress more than others depending on the data values produced by the sensors. However, F2MC [44] also reduces multimedia file formats, such as AVI, JPEG, or MP3 files, which is not the focus of Fog-DaRe. Still, it is feasible to compute new data filtering configurations into Fog-DaRe, for example, a Python Imaging Library [52], which can shrink an image file size by reducing its quality.

8. Conclusion

This paper introduces Fog-DaRe that allows the deployment of a resilient data flow for a fog-based IoT system. Fog-DaRe provides data resilience and deals with the fog resources constraints, implementing mechanisms for filtering (reducing) the data volume sent and stored by fog nodes. We also evaluate 10 data flow configurations for the Fog-DaRe system and assess each configuration in two network availability scenarios. We obtained at least 81.5% storage usage reduction and 77.7% batch transfer time reduction for long-term disconnections. However, we used only one dataset considering a smart farming scenario, and the reduction rates can vary using other dataset scenarios.

For the connection availability scenario, filtering tech-

niques cause higher packet delays and lower throughput and storage needs. Therefore, only in specific situations is it advantageous to use data filtering in the connection availability scenario. We perceive a significant increase in batch transfer times and storage usage in the scenario with temporary network unavailability as the disconnection time increases. The study also observes trade-offs between Fog-DaRe configurations regarding the filtered data details, CPU usage, RAM usage, storage usage, and the batch transfer times.

As future work, we intend to improve Fog-DaRe by incorporating more trustworthiness requirements - as redundancy, load balancing, and security techniques - into the data flow, aiming at obtaining a more robust, fast, and resilient IoT system. Additionally, we plan to evaluate our solution using other datasets and include multimedia data reduction configurations in Fog-DaRe.

References

[1] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Computer Networks 54 (15) (2010) 2787 – 2805. doi:https://doi.org/10.1016/j.comnet.2010.05.010.

[2] M. Aazam, S. Zeadally, K. A. Harras, Fog computing architecture, evaluation, and future research directions, IEEE Communications Magazine 56 (5) (2018) 46–52. doi:10.1109/MCOM.2018.1700707.

[3] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, P. A. Polakos, A comprehensive survey on fog computing: State-of-the-art and research challenges, IEEE Communications Surveys Tutorials 20 (1) (2018) 416–464. doi:10.1109/COMST.2017.2771153.

[4] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture 98 (2019) 289 – 330. doi:https://doi.org/10.1016/j.sysarc.2019.02.009.

[5] P. Ray, A survey on internet of things architectures, Journal of King Saud University - Computer and Information Sciences 30 (3) (2018) 291 – 319. doi:https://doi.org/10.1016/j.jksuci.2016.10.003.

[6] H. F. Atlam, R. J. Walters, G. B. Wills, Fog computing and the internet of things: A review, Big Data and Cognitive Computing (2018). doi:10.3390/bdcc2020010.

[7] F. M. R. Junior, C. A. Kamienski, A survey on trustworthiness for the internet of things, IEEE Access 9 (2021) 42493–42514. doi:10.1109/ACCESS.2021.3066457.

[8] J.-H. Cho, S. Xu, P. M. Hurley, M. Mackay, T. Benjamin, M. Beaumont, Stram: Measuring the trustworthiness of computer-based systems, ACM Computing Surveys 51 (2019). doi:10.1145/3277666.

[9] J. Spiegel, P. Wira, G. Hermann, A comparative experimental study of lossless compression algorithms for enhancing energy efficiency in smart meters, in: IEEE 16th International Conference on Industrial Informatics, 2018, pp. 447–452. doi:10.1109/INDIN.2018.8471921.

[10] A. Gupta, A. Bansal, V. Khanduja, Modern lossless compression techniques: Review, comparison and analysis, in: Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2017, pp. 1–8. doi:10.1109/ICECCT.2017.8117850.

[11] S. K. Routray, A. Javali, A. Sahoo, W. Semunigus, M. Pappa, Lossless compression techniques for low bandwidth io ts, in: Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2020, pp. 177–181. doi:10.1109/I-SMAC49090.2020.9243457.

[12] T. N. Gia, L. Qingqing, J. P. Queralta, H. Tenhunen, Z. Zou, T. Westerlund, Lossless compression techniques in edge computing for mission-critical applications in the iot, in: Twelfth International Conference on Mobile Computing and Ubiquitous Network (ICMU), 2019, pp. 1–2. doi:10.23919/ICMU48249.2019.9006647.

[13] R. Mahalakshmi, D. Kannan, Semantic filtering of iot data using symbolic aggregate approximation (sax), Journal of Computer Science and Applications, 2016.

[14] A. González-Vidal, J. Cuenca-Jara, A. F. Skarmeta, Iot for water management: Towards intelligent anomaly detection, in: IEEE 5th World Forum on Internet of Things (WF-IoT), 2019, pp. 858–863. doi:10.1109/WF-IoT.2019.8767190.

[15] S. A. Abdulzahra, A. K. M. Al-Qurabat, A. K. Idrees, Data reduction based on compression technique for big data in iot, in: International Conference on Emerging Smart Computing and Informatics (ESCI), 2020, pp. 103–108. doi:10.1109/ESCI48226.2020.9167636.

[16] R. Rezvani, P. Barnaghi, S. Enshaeifar, A new pattern representation method for time-series data, IEEE Transactions on Knowledge and Data Engineering (2019) 1–1doi:10.1109/TKDE.2019.2961097.

[17] N. G. S. Campos, A. R. Rocha, R. Gondim, T. L. Coelho da Silva, D. G. Gomes, Smart amp; green: An internet-of-things framework for smart irrigation, Sensors (2020). doi:10.3390/s20010190.

[18] R. Ferrando, P. Stacey, Classification of device behaviour in internet of things infrastructures: Towards distinguishing the abnormal from security threats, Association for Computing Machinery, New York, NY, USA, 2017. doi:10.1145/3109761.3109791.

[19] C. Kamienski, J.-P. Soininen, M. Taumberger, R. Dantas, A. Toscano, T. Salmon Cinotti, R. Filev Maia, A. Torre Neto, Smart water management platform: Iot-based precision irrigation for agriculture, Sensors (2019). doi:10.3390/s19020276.

[20] I. Zyrianoff, A. Heideker, D. Silva, J. Kleinschmidt, J.-P. Soininen, T. Salmon Cinotti, C. Kamienski, Architecting and deploying iot smart applications: A performance–oriented approach, Sensors (2020). doi:10.3390/s20010084.

[21] B. Omoniwa, R. Hussain, M. A. Javed, S. H. Bouk, S. A. Malik, Fog/edge computing-based iot (feciot): Architecture, applications, and research issues, IEEE Internet of Things Journal (2019) 4118–4149doi:10.1109/JIOT.2018.2875544.

[22] M. Asif-Ur-Rahman, F. Afsana, M. Mahmud, M. S. Kaiser, M. R. Ahmed, O. Kaiwartya, A. James-Taylor, Toward a heterogeneous mist, fog, and cloud-based framework for the internet of healthcare things, IEEE Internet of Things Journal (2019) 4049–4062doi:10.1109/JIOT.2018.2876088.

[23] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM (2010) 50–58doi:10.1145/1721654.1721672.

[24] K. A. Delic, On resilience of iot systems: The internet of things, no. February, Association for Computing Machinery, New York, NY, USA, 2016. doi:10.1145/2822885.

[25] V. Prokhorenko, M. Ali Babar, Architectural resilience in cloud, fog and edge systems: A survey, IEEE Access 8 (2020) 28078–28095. doi:10.1109/ACCESS.2020.2971007.

[26] D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique, E. Bartocci, A roadmap toward the resilient internet of things for cyber-physical systems, IEEE Access 7 (2019) 13260–13283. doi:10.1109/ACCESS.2019.2891969.

[27] T. N. Gia, M. Jiang, A. Rahmani, T. Westerlund, P. Liljeberg, H. Tenhunen, Fog computing in healthcare internet of things: A case study on ecg feature extraction, in: IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, 2015, pp. 356–363. doi:10.1109/CIT/IUCC/DASC/PICOM.2015.51.

[28] J. Azar, A. Makhoul, M. Barhamgi, R. Couturier, An energy efficient iot data compression approach for edge machine learning, Future Generation Computer Systems 96 (2019) 168 – 175. doi:https://doi.org/10.1016/j.future.2019.02.005.

[29] B. Negash, T. Gia, A. Anzanpour, I. Azimi, M. Jiang, T. Westerlund, A. M. Rahmani, P. Liljeberg, H. Tenhunen, Leveraging fog computing for healthcare iot., Fog Computing in the Internet of Things, 2018.doi:10.1007/978-3-319-57639-8_8.

[30] Y. Shi, G. Ding, H. Wang, H. E. Roman, S. Lu, The fog computing service for healthcare, in: 2nd International Symposium on Future In-

formation and Communication Technologies for Ubiquitous HealthCare (Ubi-HealthTech), 2015, pp. 1–5. doi:10.1109/Ubi-HealthTech.2015.7203325.

[31] N. Narendra, K. Ponnalagu, A. Ghose, S. Tamilselvam, Goal-driven context-aware data filtering in iot-based systems, in: IEEE 18th International Conference on Intelligent Transportation Systems, 2015, pp. 2172–2179. doi:10.1109/ITSC.2015.351.

[32] F. M. Ribeiro, R. Prati, R. Bianchi, C. Kamienski, A nearest neighbors based data filter for fog computing in iot smart agriculture, in: IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor), 2020, pp. 63–67. doi:10.1109/MetroAgriFor50201.2020.9277661.

[33] A. Jonathan, M. Uluyol, A. Chandra, J. Weissman, Ensuring reliability in geo-distributed edge cloud, in: Resilience Week (RWS), 2017, pp. 127–132. doi:10.1109/RWEEK.2017.8088660.

[34] Y. Harchol, A. Mushtaq, J. McCauley, A. Panda, S. Shenker, Cessna: Resilient edge-computing, MECOMM'18, Association for Computing Machinery, New York, NY, USA, 2018, p. 1–6. doi:10.1145/3229556.3229558.

[35] T. Jeong, J. Chung, J. W. Hong, S. Ha, Towards a distributed computing framework for fog, in: IEEE Fog World Congress (FWC), 2017, pp. 1–6. doi:10.1109/FWC.2017.8368528.

[36] M. Al-khafajiy, T. Baker, A. Waraich, D. Al-Jumeily, A. Hussain, Iot-fog optimal workload via fog offloading, in: IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), 2018, pp. 359–364. doi:10.1109/UCC-Companion.2018.00081.

[37] K. Fall, A delay-tolerant network architecture for challenged internets, in: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '03, Association for Computing Machinery, New York, NY, USA, 2003, p. 27–34. doi:10.1145/863955.863960.
URL https://doi.org/10.1145/863955.863960

[38] J. E. Luzuriaga, M. Zennaro, J. C. Cano, C. Calafate, P. Manzoni, A disruption tolerant architecture based on mqtt for iot applications, in: 14th IEEE Annual Consumer Communications Networking Conference (CCNC), 2017, pp. 71–76. doi:10.1109/CCNC.2017.7983084.

[39] C. Kulatunga, L. Shalloo, W. Donnelly, E. Robson, S. Ivanov, Opportunistic wireless networking for smart dairy farming, Vol. 19, 2017, pp. 16–23. doi:10.1109/MITP.2017.28.

[40] G. Castellano, F. Risso, R. Loti, Fog computing over challenged networks: A real case evaluation, in: IEEE 7th International Conference on Cloud Networking (CloudNet), 2018, pp. 1–7. doi:10.1109/CloudNet.2018.8549354.

[41] J. Moura, D. Hutchison, Fog computing systems: State of the art, research issues and future trends, with a focus on resilience, Journal of Network and Computer Applications 169 (2020) 102784. doi:https://doi.org/10.1016/j.jnca.2020.102784.

[42] S. Chandak, K. Tatwawadi, C. Wen, L. Wang, J. Aparicio Ojea, T. Weissman, Lfzip: Lossy compression of multivariate floating-point time series data via improved prediction, in: Data Compression Conference (DCC), 2020, pp. 342–351. doi:10.1109/DCC47342.2020.00042.

[43] D. Blalock, S. Madden, J. Guttag, Sprintz: Time series compression for the internet of things, Vol. 2, New York, NY, USA, 2018. doi:10.1145/3264903.

[44] Y. Fu, X. Qiu, J. Wang, F2mc: Enhancing data storage services with fog-tomulticloud hybrid computing, in: IEEE 38th International Performance Computing and Communications Conference (IPCCC), 2019, pp. 1–6. doi:10.1109/IPCCC47392.2019.8958748.

[45] A. Sinaeepourfard, J. Garcia, X. Masip-Bruin, E. Marin-Tordera, A novel architecture for efficient fog to cloud data management in smart cities, in: IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 2622–2623. doi:10.1109/ICDCS.2017.202.

[46] Chirpstack: open-source lorawan network server stack, https://www.chirpstack.io, accessed: August, 2020.

[47] Eclipse mosquitto: An open source mqtt broker, https://mosquitto.org, accessed: August, 2020.

[48] Pqueue, https://pypi.org/project/pqueue/, accessed: August, 2020.

[49] I. Zyrianoff, F. Borelli, C. Kamienski, Sense – sensor simulation environment: Uma ferramenta para geração de tráfego iot em larga escala, in: Salão de Ferramentas - Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), 2017.

[50] Fiware context broker, https://www.fiware.org/developers/, accessed: September, 2020.

[51] F. Aguilar, Press, temperature and humidity, in: Dryad, Dataset, 2014. doi:https://doi.org/10.15146/R3730R.

[52] Python pillow, https://python-pillow.org/, accessed: May, 2021.

Franklin Magalhães Ribeiro Junior received a B.S. and an M.S. degree in Computer Science from the Federal University of Sergipe (UFS), Aracaju, Brazil, in 2013 and 2015, respectively. He is currently pursuing a Ph.D. degree in Computer Science at the Federal University of ABC (UFABC). He is a lecturer of Information Technology at the Federal Institute of Maranhão (IFMA). He is a research fellow of the NUVEM Strategic Research Unit. His research interests include dependable systems, smart cities and fog computing.

Carlos Alberto Kamienski received the B.S. degree in Computer Science from the Federal University of Santa Catarina, Florianópolis, Brazil, in 1989, the M.S. degree from the State University of Campinas, Campinas, Brazil, in 1994, and the Ph.D. degree in Computer Science from the Federal University of Pernambuco, Recife, Brazil, in 2003. He is a Full Professor of Computer Science with the Federal University of ABC (UFABC), Santo André, Brazil, where he currently also holds the position of the Head of the NUVEM Strategic Research Group. His current research interests include the Internet of Things, smart agriculture, network softwarization, and fog computing.