Newcastle University

School of Computing Science

Adaptive techniques for time-critical data processing in IEC

Fawzy Mohammad Habeeb

Submitted for the degree of Doctor of Philosophy in the School of Computing Science, Newcastle University

April 2024

Abstract

Time-critical data processing presents an essential issue in the IEC since real-time decision-making and responsiveness are becoming more and more important in many different IEC applications, specifically those involving vital infrastructure, transportation systems, industrial automation, and healthcare monitoring. These applications require low latency, high bandwidth, the sustainability of devices, and ensuring data integrity to work effectively and reliably.

Edge computing has become increasingly popular as an addition to cloud computing, particularly for applications such as industrial control systems that demand guarantees for timely communication. Although edge computing allows for the rapid analysis of data streamed from the Internet of Things (IoT) devices, these devices often do not have the computational power and bandwidth necessary to ensure satisfactory performance for applications that are sensitive to timing. Therefore, developing a dynamic, distributed approach to manage time-critical IoT data streams efficiently across the IEC continuum is necessary.

Numerous scenarios, such as flood control and crisis management, employ thousands of energy-aware sensors. These sensors continue monitoring their environment all the time, gathering vital information that helps them make important decisions. However, since they run on batteries, energy efficiency is critical to their long-term viability. Finding adaptive, time-sensitive, cost-effective solutions that maximise their power usage is essential to extending their lifespan.

IoT has quickly become a transformative model for linking devices to gather data, share information, and process data efficiently. Faults within IoT systems can arise in multiple scenarios and manifest differently. Understanding and handling these faults is crucial for improving the integrity and reliability of real-time data for making informed decisions and maintaining the effectiveness of IoT applications. Monitoring real-time data streams on a constant basis for abnormalities, faults, or inconsistencies that can be caused by environmental conditions, communication problems, or malfunctioning

sensors is necessary to manage data quality and failure.

To handle bandwidth optimisation, energy enhancement, data quality monitoring, and healing, this thesis presents multilateral research towards adaptive techniques for the optimisation of time-critical data processing in IEC.

The following are the thesis's main contributions:

- A novel distributed and QoS-based multi-level queue traffic scheduling system that can undertake semi-automatic bandwidth slicing to process time-critical incoming traffic in IEC environments.

- A novel approach for optimising energy-efficient IoT devices for time-sensitive data streams using Reinforcement Learning(RL), which optimises energy efficiency while ensuring timely data delivery for critical applications.

- A dynamic framework based on multi-agent RL for detecting and handling faults in an IoT-edge environment, which optimises the real-time data quality to ensure its integrity to work effectively and reliably.

Declaration

I declare that this thesis is my original work unless explicitly stated otherwise. None of the material included in this thesis has been previously submitted for any degree or qualification at Newcastle University or any other academic institution.

Fawzy Mohammad Habeeb

April 2024

Publications

Published

1. Alwasel, K., Jha, D.N., **Habeeb, F**., Demirbaga, U., Rana, O., Baker, T., Dustdar, S., Villari, M., James, P., Solaiman, E. and Ranjan, R., 2021. IoTSim-Osmosis: A framework for modeling and simulating IoT applications over an edge-cloud continuum. Journal of Systems Architecture, 116, p.101956.

2. **Habeeb, F**., Alwasel, K., Noor, A., Jha, D.N., Alqattan, D., Li, Y., Aujla, G.S., Szydlo, T. and Ranjan, R., 2022. Dynamic Bandwidth Slicing for Time-Critical IoT Data Streams in the Edge-Cloud Continuum. IEEE Transactions on Industrial Informatics.

3. Szydlo, T., Szabala, A., Kordiumov, N., Siuzdak, K., Wolski, L., Alwasel, K., **Habeeb, F**. and Ranjan, R., 2022. IoTSim-Osmosis-RES: Towards autonomic renewable energy-aware osmotic computing. Software: Practice and Experience.

4. **Habeeb, F**., Szydlo, T., Kowalski, L., Noor, A., Thakker, D., Morgan, G. and Ranjan, R., 2022. Dynamic Data Streams for Time-Critical IoT Systems in Energy-Aware IoT Devices Using Reinforcement Learning. Sensors, 22(6), p.2375.

5. Alqattan, D., Ojha, V., **Habib, F**., Noor, A., Morgan, G. and Ranjan, R., 2024. Modular neural network for Edge-based Detection of early-stage IoT Botnet. High-Confidence Computing, p.100230.

Acknowledgements

First and foremost, I express my gratitude to Allah for His blessings, which have enabled me to complete this thesis. I am thankful for the good health and well-being that were essential throughout this journey. I consider this work a form of worship to Allah, and I praise Him for His guidance and support. All praise and thanks belong to Allah, both at the beginning and end. As commanded in the Quran, saying "So hold that which I have given you and be of the grateful." Surah Al-A'raf:144. I am grateful to Allah and express my thanks to people as well.

I extend my sincere appreciation to my thesis supervisor, Professor Rajiv Ranjan, Chair Professor in Computing Science and Internet of Things at Newcastle University. His door was always open for guidance and support whenever I faced challenges or had questions about my research. Without his assistance, this work would not have been possible.

I am grateful to my colleagues and lab mates for their support and collaboration, Ayman Noor, Devki Nandan Jha, Duaa AlQattan, Sultan Altarrazi, Nipun Balan, and Yinhao Li.

Special thanks to the government of Saudi Arabia, represented by the University of Jeddah and the Royal Embassy of Saudi Arabia Cultural Bureau in London, for funding my PhD. Their support was crucial for the completion of this journey.

I also acknowledge Professor Graham Morgan and Dr. Tomasz Szydlo as my second and third supervisors, respectively. Their valuable comments and guidance were instrumental in shaping this thesis.

My deepest gratitude goes to my parents. Firstly, my father (Mohammad Habeeb) exemplified the essence of family care, dedicating himself to his children and wife wholeheartedly. His support extended beyond financial assistance, nurturing strong spiritual bonds within our family. My mother (Najlah Nabolsi) embodies beauty in my eyes. She epitomises patience, kindness, generosity, and selflessness as a mother who wholeheartedly believes in me.

Also, thanks to my grandfather (Hassan Habeeb) and my grandmother (Fatmah Abualtaher) they have been pivotal figures in shaping our family's legacy. Their wisdom and values were the bedrock upon which our family was built. Even as they passed away during my PhD journey may Allah forgive them and have mercy on them. Their memory and teachings continue to inspire and guide me. Their lives were a testament to resilience, love, and the profound impact one leaves on this world, engraving a legacy that transcends time.

I am indebted to my wife, Walaa Abualtaher, for her unwavering support and encouragement throughout my studies and the process of researching and writing this thesis. Her presence made this accomplishment possible.

Lastly, I am thankful to my dear brother, Abdullah Habeeb, whose love, companionship, and cooperation have been invaluable to me. Thank you all.

Contents

List of Figures

List of Tables

1

Introduction

Contents

Introduction

Time-critical data processing is a computing domain that concentrates on examining and handling data within a specified time frame to fulfil the needs of particular applications or systems. This notion holds particular significance in situations where prompt decision-making is crucial, and any delays in processing could result in substantial repercussions, including system malfunctions, safety risks, or even threats to human life. Several time-critical applications and services, including industrial control systems, energy management in smart home environments, automotive, flood monitoring systems, and more, are using this concept [2], [3].

The revolution in computing architecture has been driven by the continuing expansion of innovative sensing devices and computing methods and the ever-increasing need for time-critical applications on the edge and in the cloud [4], [5]. Also in this context, an enormous number of devices are capable of receiving, processing, and sending data across the Internet or different types of networks to other systems and devices. Thanks to the concept known as IoT, data can be gathered from various sources in the real world [6].

Considering the edge cloud often handles the majority of computational data processing, an adaptable link between all parties is essential to move the edge as near to the cloud as feasible and the other way around. In collaboration with the cloud, edge computing is stretching the limitations of conventional systems of cloud computing, providing capabilities such as storage, low-latency service execution, and effective data processing. Therefore, this leads us to a term called the IEC continuum, a system comprising both edge and cloud components, and IoT resources that are created by the smooth, intelligent, and dynamic interaction of these components, leading to a paradigm change in computing. The IEC continuum embodies a decentralised computing structure that seamlessly merges edge computing with cloud computing, providing a scalable, adaptable, and effective framework for promptly processing time-sensitive data across diverse applications. This continuum proves particularly advantageous and pertinent in addressing the intricate challenges linked to various applications of time-critical data processing [7], [8].

The fundamental elements of connecting devices together to share data are known as network systems. Aligned with the IoT paradigm, applications based on IoT utilise various integrated ecosystems, including edge and cloud computing, as well as software-defined networking (SDN) and software-defined wide area networks (SD-WAN) [9]. These ecosystems provide extensive capabilities for processing and transmitting data in accordance with the quality of service (QoS) requirements of IoT applications. With its associated industrial ecosystems, the IoT paradigm represents a significant advancement in technological development [10].

Network systems in IEC should be consistently improved and developed to achieve the requirements of networking for modern applications and optimise data transmission.

Furthermore, IoT devices are integral to time-critical systems, where obtaining real-time data processing results promptly is crucial. Examples of such systems include various solutions utilised during natural disasters, such as fires and floods. The essential aspect of such systems is the processing of current, non-delayed data from sensors installed in IoT devices. To accomplish this, devices need to be capable of transmitting a real-time data stream with the required specifications. However, transferring a substantial amount of data from sensors requires a significant amount of energy to perform the measurements and then send the data to the edge and clouds for processing [11].

Furthermore, the IoT has rapidly emerged as a paradigm for connecting devices for data collection, sharing, and effective processing. IoT faults can happen in various forms and conditions. Faults can arise due to hardware malfunctions, software glitches, environmental factors, network issues, or human errors. In the realm of IoT, these faults can vary from minor inconveniences to major disruptions that affect the entire system. They have the potential to negatively affect performance and compromise data integrity [12]. Comprehending IoT faults aims to enhance IoT systems' stability, efficiency, data reliability, and integrity. It encourages us to conceive resilient methodologies and solutions for detecting, and healing faults.

1.1 Project Motivation

The drive behind this study is to explore the core obstacles involved in enhancing the efficiency of real-time data processing within IEC systems. Implementing, developing, and managing optimisation techniques and strategies for processing the real-time data in the IEC is this research's main goal. The efficiency of real-time data processing relies on the capability to adjust dynamically to evolving environmental conditions, data, and device conditions while adhering to excellent performance standards. Obtaining this goal, however, poses new challenges.

Figure 1.1: Time-Critical applications

Time-sensitive data processing is crucial across various sectors, each presenting distinct challenges that require innovative solutions. Figure 1.1 illustrates several time-sensitive applications. In smart healthcare, real-time monitoring systems track patient vital signs like blood oxygen levels and heart rate to promptly detect critical changes in a patient's condition. Wearable technology and IoT devices enable continuous data collection, allowing healthcare professionals to intervene swiftly, particularly in emergencies such as strokes or heart attacks. Nevertheless, this application encounters

hurdles such as guaranteeing the confidentiality and protection of sensitive health information, alongside the requirement for constant power availability and dependable network connections to sustain ongoing monitoring and data transfer [13].

Automotive vehicles rely on time-sensitive data processing to interpret sensor data from their surroundings, including other vehicles, road conditions, or pedestrians, making instantaneous decisions to navigate the roads safely. These systems must handle vast data volumes from cameras, radar sensors, and Light Detection and Ranging (LiDAR) necessitating substantial advanced algorithms and computational power to achieve real-time decision-making. Challenges in this context involve the necessity for fault tolerance and redundancy to avert system breakdowns, as well as adapting to fluctuating network conditions that may impact the vehicle's communication with other vehicles and external systems. [14].

Environmental monitoring applications, such as air pollution and flood monitoring, deploy sensor networks to gather data on rainfall intensity, water levels, and air quality parameters. This data facilitates predictive models to anticipate potential disasters and alert the public and authorities in time to take preventive actions. In these scenarios, the challenges encountered typically revolve around setting up and managing sensor networks in remote or challenging environments, guaranteeing the durability and dependability of the sensors, and creating algorithms that can effectively analyse environmental data to issue timely alerts [15].

The essential role of time-sensitive data processing across these varied applications underscores its significance in contemporary society. Whether enhancing patient outcomes, ensuring the safety of mitigating environmental disasters, or autonomous vehicles, the capability to respond and process data in real-time is paramount.

The IEC continuum is fundamentally reshaping the implementation of time-critical data processing across diverse sectors. By capitalising on the advantages offered by both edge and cloud computing approaches, the IEC continuum provides a solution that tackles the latency, resource management, reliability, scalability, and security issues inherent in time-critical applications. This distributed computing strategy not only boosts the efficiency and efficacy of these applications but also introduces new opportunities for innovation and advancement in time-sensitive data processing [16].

1.1.1 Challenges

Finding the appropriate solution to optimise real-time data processing in IEC systems involves many challenges. Here are some of the main challenges addressed in this thesis:

Heterogeneous network: The heterogeneity of communication components in the IEC infrastructure makes it difficult to optimise data flow and bandwidth usage to mitigate transmission mismatches in the network. It occurs when incoming data flows exceed the transmission capabilities of the network. This lack of balance can arise from unstable network connections and differences in data volume, which leads to high latency, data loss, insufficient use of network resources, and deterioration in system performance.

Heterogeneous computing: Variations in computing power, limits of resources, and system configurations between different data-processing devices and systems make it challenging to alleviate the processing inconsistency in the IEC heterogeneous architecture. This situation happens when one or more edge or cloud computational resources fail to react to incoming requests efficiently and quickly.

Battery consumption: Processing and transferring data between IoT, edge, and cloud, especially in time-critical systems, consumes energy. Battery consumption can be affected by several factors, including the volume of data generated, the duration and speed of its generation, and the data transmission process. Also, its management is complex, particularly for IoT and edge devices that operate on batteries of limited capacity and, after a certain time, need to be recharged. These devices need to maintain the battery's lifetime for as long as possible because, in some cases, it is difficult to recharge the batteries. For example, sensors in the mountains or flood streams and rivers. Therefore, improving and maintaining battery consumption on the IEC continuum systems is a challenge.

Faults: The IEC environments are vulnerable to faults due to their complexity, i.e., the large number, diversity, and heterogeneity of devices. Especially faults that occur in the network when transmitting data between devices and in IoT and edge devices when processing data. Due to hardware failures, environmental conditions, software bugs,

human mistakes, or network problems. This may lead to data loss, low response times, and low throughput. Therefore, to optimise the data quality in the IEC, managing failures adaptively, i.e., identifying, detecting, and handling faults, in this environment remains a challenge.

This PhD project aims to find optimal real-time data processing solutions in the IEC continuum field while considering the aforementioned challenges. Particularly, this PhD thesis is led by the following research questions:

- What is the best way to satisfy the latency constraints along with accelerating data transmissions for IoT safety-critical applications in the IEC Continuum systems?

- How to do real-time data generation on the IoT while optimising the energy of resource-constrained devices in time-critical systems on the IEC continuum?

- How can we identify, detect, and handle faults in IoT and edge devices on the IEC continuum systems so the data quality is not compromised?

1.2 Contributions

Optimising real-time data processing in the IEC continuum field includes several areas this thesis focuses on (a) network optimisation in IoT, edge, and cloud heterogeneous environments; (b) battery optimisation in IoT and edge devices; and (c) data quality monitoring and healing in IoT and edge heterogeneous environments. The main contributions of this thesis are summarised below:

- To address the first question, we introduce an innovative distributed IoT framework. This framework is founded on multi-level network-host queuing mechanisms, prioritisation, and SDN network traffic slicing. We propose a new heuristic auto-adaptation algorithm to dynamically adjust bandwidth slicing based on the observed network utilisation for each priority, as demonstrated in Chapter 3. The system aims to optimise network and host resources within the edge-cloud

continuum, reducing queuing delays, and maximising QoS for IoT applications sensitive to high latency as much as possible.

- To address the second question mentioned earlier, we introduce a solution based on reinforcement learning (RL), called dynamic data streams, designed for energy-conscious IoT devices in time-sensitive IoT systems. This proposed mechanism can adapt the data transmission rate according to the availability of renewable energy resources, ensuring consistent data collection while also considering the lifespan of the sensor battery (illustrated in Chapter 4 later).

- To solve the third question aforementioned, we introduce a novel framework for an adaptive multi-agent system for fault detection and handling in IoT-edge heterogeneous systems. We utilise Multi-Agent Reinforcement Learning (MARL), where each agent observes and learns from the IoT device. Then, interaction among the various agents is developed to improve each agent's learning by taking into account the knowledge of the agents at the different devices in the network. The aforementioned method enables the detection of abnormalities throughout the whole system architecture (demonstrated in Chapter 5 later on).

1.3 Thesis Structure

Figure 1.2 explains the thesis's structure. The chapters were derived from some publications that I released while pursuing my PhD. The subsequent chapters of the thesis are organised as follows:

- Chapter 2 illustrates the background of optimisation, adaptation, and the IoT Edge Cloud (IEC) continuum field and discusses related work on different techniques and methods for improving data processing and transition, battery consumption optimisation, fault detection, and handling.

- Chapter 3 Introduces an innovative distributed and QoS oriented multi-level queue traffic scheduling system capable of semi-automated bandwidth allocation to handle time-critical incoming traffic within edge-cloud environments. This chapter has been derived from:

  - **Habeeb, F**., Alwasel, K., Noor, A., Jha, D.N., Alqattan, D., Li, Y., Aujla, G.S., Szydlo, T. and Ranjan, R., 2022. Dynamic Bandwidth Slicing for Time-Critical IoT Data Streams in the Edge-Cloud Continuum. IEEE Transactions on Industrial Informatics.

- Chapter 4 presents a dynamic data stream based on reinforcement learning (RL) in energy-conscious IoT devices deployed for time-critical IoT systems. This chapter has been derived from:

  - **Habeeb, F**., Szydlo, T., Kowalski, L., Noor, A., Thakker, D., Morgan, G. and Ranjan, R., 2022. Dynamic Data Streams for Time-Critical IoT Systems in Energy-Aware IoT Devices Using Reinforcement Learning. Sensors, 22(6), p.2375

- Chapter 5 presents a novel framework for detection, identification, handling faults and failures, and optimising the data processing performance dynamically based on a multi-agent system in an IoT system.

- Chapter 6 summarises all the work that was completed in this thesis and provides some ideas for future work.

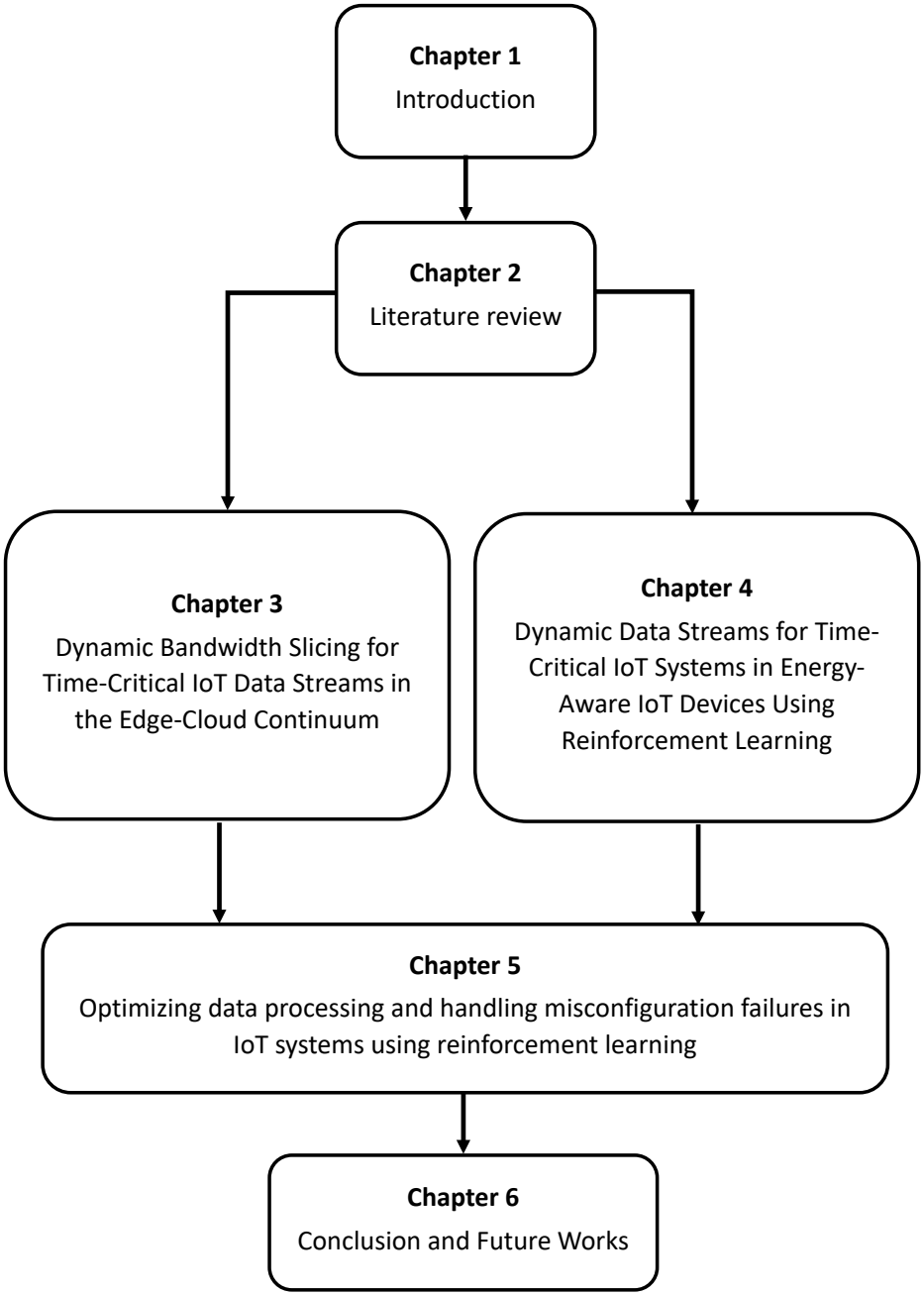


Figure 1.2: Thesis outline.

2

Literature review

Contents

Summary

This chapter provides background information on several key topics relevant to the overarching theme, including an overview of IEC, IoT, edge computing, and cloud computing environments. Next, we discussed optimisation and adaptation in IEC. Moreover, in the IEC continuum, we used network, energy optimisation, fault management, and data quality techniques. A central focus of this thesis is to tackle the challenges associated with optimising IEC devices, servers, and data quality and managing failures and anomalies in IEC infrastructure. At the end of this chapter, we discussed the existing research gaps while briefly outlining how this thesis endeavours to bridge these gaps.

2.1 IoT Edge Cloud (IEC) Continuum

The IEC Continuum describes a network framework that combines IoT, edge computing, and cloud computing into a cohesive and interconnected system. This model is distinguished by the cooperative interaction among diverse components, for example, IoT devices, edge computing devices, and centralised cloud servers, working together to handle and monitor the data generated by connected devices [16].

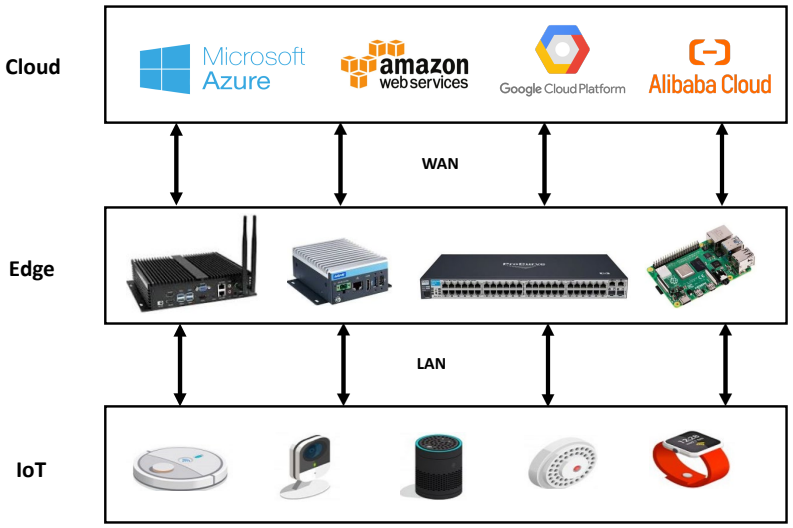


Figure 2.1: An overview of the infrastructure of the IEC continuum.

Figure 2.1 shows the components and data stream paths in the IEC infrastructure. Starting with IoT, which consists of smart devices, sensors, etc., that collect data from

the surroundings and send it through an IoT gateway to edge devices via LAN network, using multiple protocols such as Bluetooth, Zigbee, Wi-Fi, etc., which depends on range, power consumption, and bandwidth [17]. Next, the edge, which consists of Raspberry Pi, UDOO boards, switches, etc., is near to the IoT devices, providing processing and storage for the data sent by the IoT quickly and with low latency in the network [18]. Then it is sent through the edge gateways through the WAN via the internet to the servers in the cloud. The cloud layer offers a broader range of computing, analytical, and storage services compared to the edge. It provides resources with varying characteristics at different costs, depending on their utilisation. There are many providers of these services, such as Microsoft Azure, Amazon, Alibaba Cloud, etc [19].

2.1.1 IoT

The IoT represents a pioneering concept that leverages sophisticated wireless communication technologies to link a wide array of entities. These include sensors, actuators, mobile phones, and vehicles, all communicating with one another to fulfil specific service objectives [20, 21]. The idea of IoT was initially introduced at the MIT Auto-ID labs by Kevin Ashton in 1999[22]. The term ”IoT” encompasses a broad spectrum of meanings, referring to a global network that connects various objects surrounded by human environments through cutting-edge communication technologies. Additionally, IoT signifies the concept of physical objects (”things”) that have the capability to interact and cooperate with each other to accomplish shared aims [23]. The International Telecommunication Union (ITU) in 2005, released a document regarding the IoT, officially endorsing the IoT movement. This report highlighted the onset of a ubiquitous era of IoT communications, where the active exchange of information among all objects worldwide via networks became possible [24].

Based on the most recent data, there are around 17.08 billion interconnected IoT devices in 2024. This number is projected to nearly double, reaching 29.42 billion by the year 2030 IoT devices, a vast network demanding meticulous organisation while taking into account critical aspects such as energy efficiency, mobility, reliability, network coverage, link capacity, and the cost of devices[25, 26]. IoT can be described as

a pervasive system designed to bridge "things" in the tangible, physical world with the intangible, cyber, or virtual realm. In the context of IoT, the term "things" refers to a diverse array of physical objects with tangible forms, each capable of responding to specific real-world conditions [27]. On the other hand, the cyber world encompasses services, digital actions, and entities (such as applications), essentially representing the digital functionalities that physical objects can offer to perform particular tasks. Figure 2.2 illustrates the IoT network along with a selection of its applications, highlighting actuators and sensors as the primary physical components of an IoT setup. Each of these elements is characterised by unique properties, as detailed further in the discussion [28]:

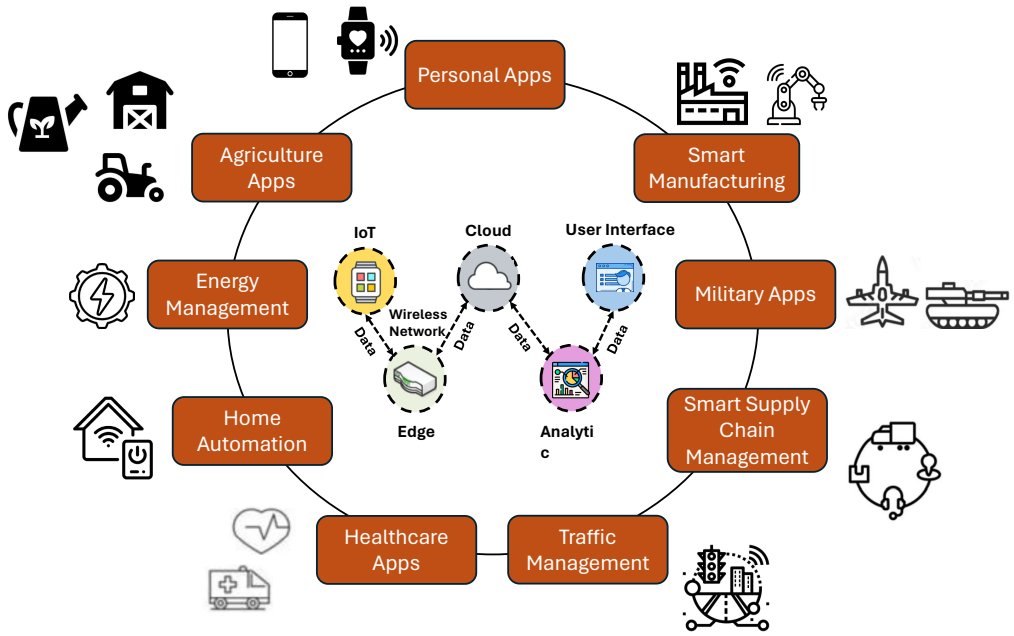


Figure 2.2: Edge computing architecture

A- Sensors: a sensor serves as a device capable of detecting various inputs from its physical environment around it. These inputs may include factors like smoke, humidity, sound, light, heat, touch, motion, pressure, moisture, or numerous other environmental phenomena. Typically, the output generated is a signal that can either be presented in a human-readable format directly at the sensor location or transmitted electronically through a network for further analysis or processing.

B- Actuators: actuators serve as essential components in automation and control systems, translating collected data into actionable commands to improve performance

in automotive applications. These actuators are typically mechanical or electronic devices, such as switches, responsible for executing commands and directives by initiating appropriate actions on physical objects within the surrounding environment.

Those IoT applications are integrated using several wireless communication technologies, such as, RFID, LoWPAN, Bluetooth, Wi-Fi, and ZigBee [29].

2.1.1.1 IoT Applications

IoT will influence numerous application domains, as depicted in Figure 2.2. This section categorises IoT applications based on their respective domains as follows:

Automotive: Time-sensitive IoT applications are important in the automotive industry for improving vehicle safety, efficiency, and user experience. Continuous monitoring of vehicle performance and condition, including, for example, tyre pressure, engine health, and fuel levels, enables immediate alerts and proactive maintenance measures. Moreover, IoT-equipped vehicles communicate with each other to prevent accidents, optimise traffic flow, and offer real-time navigation adjustments based on current traffic conditions, thereby enhancing road safety and minimising congestion [30].

Environmental Monitoring: Time-critical IoT applications are transforming environmental monitoring, this enables immediate monitoring, analysis, and response to diverse environmental conditions and potential hazards in real-time. These applications leverage a network of interconnected sensors deployed across diverse ecosystems and urban areas to collect data on water quality, air quality, wildlife activities, soil conditions, and more. This real-time data collection and analysis facilitates immediate actions and making well-informed decisions in reaction to alterations in the environment and unforeseen circumstances [31].

Flood Monitoring: IoT applications in flood monitoring utilise real-time data collection from sensors distributed across rivers, dams, and flood-prone areas to detect early signs of flooding. This immediate data analysis enables the rapid dissemination of warnings to communities and emergency services, facilitating timely evacuations and preparations. The integration of IoT in flood risk management transforms traditional reactive approaches into proactive measures, significantly reducing the impact of floods

through early detection and swift response strategies [32].

Healthcare: In the healthcare domain, time-critical IoT applications are revolutionising patient care, emergency response, and overall medical services. Real-time health monitoring systems, powered by IoT devices, continuously track vital signs and other health indicators, enabling instant detection of anomalies or critical conditions. This immediacy ensures that healthcare providers can respond to patients' needs swiftly, whether for routine monitoring or emergencies. For instance, wearable IoT devices such as heart rate monitors, glucose monitors, and smart patches transmit health data in real time to healthcare professionals, allowing for immediate analysis and intervention. In emergencies, such as heart attacks or diabetic shocks, the devices can automatically alert medical services, providing them with crucial information even before they arrive on the scene [33].

Personal Appliances: In personal IoT applications, sensors closely associated with individuals deliver real-time data for immediate action. These encompass wearable health monitors and fitness devices, offering instant feedback and alerts. Smartphones act as vital communication hubs, facilitating seamless data transmission for immediate analysis or emergency response, catering to industries like healthcare, fitness, and personal safety with a focus on low-latency solutions for wearable technology[34].

Home Automation: IoT integration in home automation systems enables real-time monitoring and control over household devices via a private network. This instantaneity allows for critical applications such as emergency alerts, immediate temperature adjustments for safety, and security breaches, ensuring swift action to maintain safety and comfort [35].

Smart Manufacturing: IoT in manufacturing emphasises real-time monitoring and control, crucial for time-sensitive processes. Immediate data analysis and feedback allow for rapid adjustments in manufacturing lines, enhancing efficiency, reducing downtime, and ensuring quality control in critical production processes [36].

Smart Supply Chain Management: In logistics, IoT applications prioritise real-time tracking and management, ensuring timely updates on shipment status and location. This immediacy improves efficiency, reduces delays, and enhances the reliability of

supply chains by providing instant information for decision-making [37].

2.1.2 Edge computing

The concept of situating computer resources close to data origins is not a novel idea [38]. The phrase "edge computing" was first coined in 2002, indicating a business-driven need to Move applications from cloud data centres to the network edge. By 2004, the term had evolved to describe a framework where programme methods and related data were distributed to the network's edge, aiming to boost system performance [39].

Hence, edge computing arises as a model that enhances cloud computing architectures by processing data close to its source, particularly at the network periphery. This approach facilitates the deployment of computing and storage resources near the data source, primarily at the network edge, resulting in improved efficiency and performance. Edge computing has evolved to address various challenges encountered in cloud computing by providing flexible resources directly to users at the network edge, a capability not initially available with cloud computing, which centralised its resources in data centres at the network core. The evolution of computing has transitioned from individual computers managing various tasks to centralised services and applications hosted in cloud data centres, marking a significant advancement over the past decade [40].

Recent technological innovations, including advanced home connection devices, high-capacity mobile devices, wireless networks, and growing concerns over reliability, privacy, and user autonomy, underscore the necessity of managing computing applications, data, and services at the internet's outermost layer (the "edge") rather than its central points (the "core") [41].

Driven by the IoT, the imminent landscape of digital connectivity anticipates the integration of nearly all electrical devices into the IoT framework. These devices, ranging from air quality sensors and LED lighting to streetlights and internet-enabled microwave ovens, will simultaneously serve as generators and recipients of data. It's projected that, in the coming years, the proliferation of these edge devices will reach the billions, collectively producing an astronomical volume of raw data that will overwhelm the traditional capabilities of cloud computing infrastructures. Consequently,

a substantial portion of IoT-generated data is likely to be processed locally at the network edge rather than being sent to centralised cloud data centres [42].

These driving forces collectively underscore the necessity for a paradigm shift towards edge computing, where data is processed closer to its point of origin. This approach not only alleviates the burden on cloud infrastructure but also addresses the unique requirements of the burgeoning IoT landscape, oping the way for a more efficient, secure system, reducing response times, alleviating network congestion, and enhancing the efficiency of data processing.

2.1.2.1 Architecture

The fundamental concept of edge computing revolves around conducting computational operations at the network's periphery, near the data's source. These resources extend from the network and computing assets shared among end-users to fog nodes and cloud data centres. This ecosystem encompasses interconnected sensors and devices within the IoT layer, communicating and sharing data through sophisticated network infrastructures. Data processing takes place at various tiers depending on the application's specific requirements, Structured into three main layers: IoT, edge, and cloud, as depicted in a typical edge computing architecture Figure 2.3 [43].

Within this architecture, the IoT layer encompasses millions of sensors and devices continuously generating data, sharing vital information through sophisticated communication networks, and overseeing essential functions within smart infrastructures. These IoT devices and sensors are the primary consumers of edge computing services. While IoT and edge are developing rapidly and independently, the integration of edge can significantly enhance IoT by addressing several critical challenges and boosting performance. Typically, IoT devices stand to gain from the substantial computational power and storage capabilities available across the edge and cloud layers [44].

However, edge presents additional benefits for IoT beyond what cloud computing can offer, despite having somewhat lesser capacity and storage. The primary requirement for IoT applications is not immense computational power or extensive storage but rather swift response times. Edge computing is capable of providing adequate computational strength, sufficient storage, and the rapid response times needed by IoT

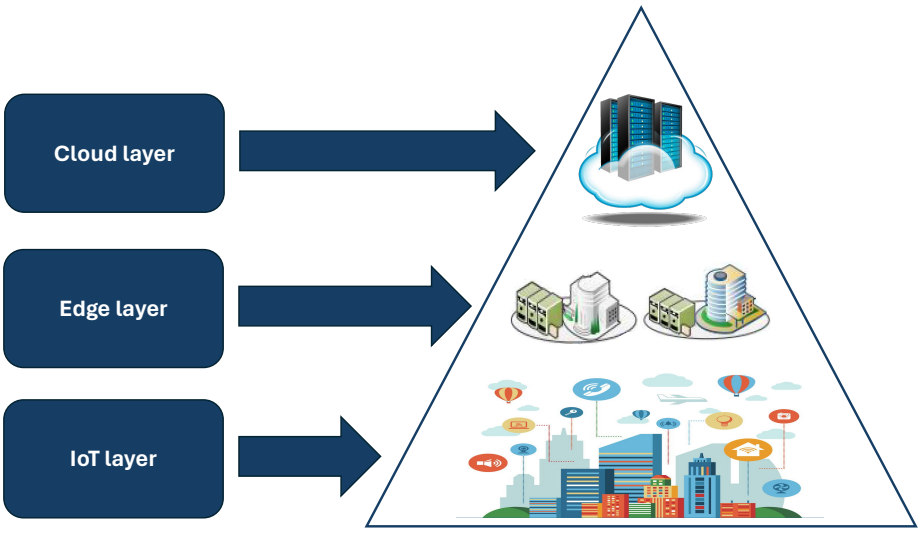


Figure 2.3: Edge computing architecture

applications. Moreover, edge computing benefits from IoT through the expansion of the edge infrastructure to manage distributed computing nodes, With IoT devices functioning as edge nodes for service delivery, the integration between IoT and edge computing is anticipated to expand, creating a growing interdependence between them [45].

2.1.2.2 Characteristics

IoT and edge computing share several similarities, particularly in relation to the IoT layer. The primary aim of edge computing is to extend the capabilities of cloud computing to the edge of the network. This approach positions computational and storage resources near end-users, aiming to minimise service latency and conserve network bandwidth for applications sensitive to delays. Unlike the centralised structure of cloud computing, the edge computing framework is characterised by its distributed, hierarchical, and decentralised designs, with its services located closer to end users. Compared to the vast resources available in cloud computing, the resources of edge computing (including computing power, communication capabilities, and storage) are relatively limited [46]. The characteristics of all layers is demonstrated in Table 2.4.

2.1.3 Cloud computing

Cloud computing, an internet-based model of computing, enables the sharing of pro-

Characteristic	IoT	Edge	Cloud
Components	Physical devices	Edge nodes	Virtual resources
Deployment	Distributed	Distributed	Centralised
Data	Source	Process	Process
Computational	Limited	Limited	Unlimited
Storage	Very limited	Limited	Unlimited
Distance to data source	The source	The nearest	Far
Nodes count	The largest	Very large	Small
Location awareness	Aware	Aware	Not aware
Response time	No response time	The fastest	Slow

Figure 2.4: Characteristic of IoT, edge, and cloud computing table

cessing power, storage resources, and data across various devices (for example, computers, laptops, tablets and smartphones) on a demand basis. This model provides extensive and Immediate access to various computing resources like networks, storage, servers, and applications as required. It provides users and businesses with the ability to process and store data in third-party data centres, significantly reducing infrastructure costs for corporations. Cloud computing also enhances the speed of application deployment, improves manageability, and requires less maintenance, while allowing for the rapid and flexible adjustment of resources (software, hardware, network, and services) to meet business demands [47]. The rise of cloud computing has been driven by factors such as high-capacity networks, affordable storage devices and computing power, along with a growing inclination towards hardware virtualisation and autonomic computing. The advantages of adopting cloud computing are numerous, including dynamic scaling, reduced capital expenditures, high availability, ease of management, resource sharing, cost-effective services, superior performance, accessibility, enhanced productivity, reliability, increased mobility, and environmental friendliness. These benefits have made cloud computing a highly sought-after service

[48]. Examples of leading cloud service providers include Microsoft Azure [49], Google Cloud [50], and Amazon AWS [51]).

2.2 Optimisation and Adaptation in IEC

In the realm of computing, optimisation involves altering a system to enhance its performance or reduce its resource consumption, ranging from low-level tasks such as circuit development and machine code customisation for specific architectures to high-level activities including algorithm design and implementation. For example, software may be optimised to increase its speed, lower its memory usage, or decrease its energy consumption. Optimising applications within the IEC continuum addresses the unique challenges and needs of the IEC's heterogeneous and dispersed architecture, which spans all computing resources, from IoT and edge to centralised cloud infrastructures. Optimisation in this context aims to achieve equilibrium in data processing, communication, energy, and data integrity needs between the IoT, edge, and cloud components [52].

Adaptation, on the other hand, is a crucial concept that focuses on the ability of systems, applications, or processes to dynamically adjust to changing conditions or requirements. This adaptability is particularly important in today's digital landscape, where the variability of user demands, resource availability, and system conditions can significantly impact performance, efficiency, and user satisfaction. The need for adaptation arises from several key factors, such as dynamic environments, resource optimisation, enhanced user experience, fault tolerance, resilience, as well as security and compliance. Adaptive methods, which include a collection of practices or instruments designed to allow applications to actively adjust to changing circumstances, play a critical role in this optimisation process. These methods can include strategies like the automatic adjustment of resource levels, the dynamic allocation of computing resources, and mechanisms for self-handling [53].

In the context of the IEC framework, adaptive techniques refer to the methods and strategies employed to enable IEC systems and devices to adjust to changing conditions or requirements. These techniques are crucial for improving the efficiency, perfor-

mance, and reliability of IoT systems in various environments, making them especially beneficial in unpredictable and changing circumstances. The relationship between optimisation and adaptation is inherently synergistic; while optimisation seeks to find the best configuration for a given scenario, adaptation ensures that the system remains optimal or near-optimal as the scenario changes. This dynamic optimisation process involves continuously monitoring the environment, predicting future states, and making preemptive adjustments to maintain or improve performance, efficiency, and reliability. The benefits of integrating adaptation into optimisation strategies include continuous optimisation, proactive problem-solving, scalability and flexibility, and sustainability [54].

In sections 2.3 and 2.4, we present several techniques for bandwidth and energy optimisation, fault management, and data quality, demonstrating how adaptive methods enhance system robustness, and fault handling by proactively addressing new abnormalities. The shift towards adaptive techniques is motivated by the necessity for systems to dynamically adjust to evolving situations and environments, ensuring that systems are effective, adaptable, and quick to respond to changing conditions.

2.3 Network management technique in the IEC environment

Networks play a crucial role, serving as the backbone that connects various components within the IEC continuum [55]. These networks facilitate smooth data transmission among IoT devices situated at the network's edge and the central cloud infrastructure, facilitating real-time data analysis, decision-making, and control and ensuring the scalability, reliability, and efficiency of IEC systems. Starts by describing SDN and its architecture as compared to traditional networks. Then, WAN and SD-WAN. Next, bandwidth slicing and several scheduling algorithms, including multi-level queues, FCFS, and SJN. Our focus is on optimising data flow and bandwidth utilisation.

2.3.1 Software Defined Networking (SDN)

SDN is an approach to networking that employs software-based controllers or application programming interfaces (APIs) to manage traffic flow within the network and communicate with the underlying hardware infrastructure [56, 57]. In contrast to traditional networking, where control (the system that decides where to send packets) and data planes (the system that actually forwards packets to the selected destination) are integrated within networking devices like switches and routers, SDN relocates network control logic from these devices to SDN controllers. This shift allows network administrators to oversee network services by abstracting lower-level functionalities, essentially programming the network to efficiently direct traffic. SDN strives to facilitate the dynamic configuration of networks and surmount the constraints of traditional network infrastructure [58]. By adopting this approach, networks become more resilient, straightforward, and adaptable to changes, as network control is centralised instead of relying on complex, distributed control mechanisms.

2.3.2 WAN and Software Defined Networking (SD-WAN)

Wide Area Networks (WANs) serve as the foundational communication infrastructures linking geographically dispersed systems and devices into a unified network. While traditional WANs have facilitated the interconnection of distributed systems, they demonstrate drawbacks like the absence of adaptable routing behaviour, uneven distribution of loads, reliance on intricate network protocols, absence of prioritisation, and the necessity for specialised hardware are among the challenges. These constraints limit the management and deployment of traditional WANs in environments with data-intensive applications. For the realisation of fully distributed cloud data centres, ensuring optimal resource management and efficiency in contemporary data-driven applications such as smart energy clouds, content delivery networks, and distributed gaming platforms, it is imperative to incorporate the WAN into a comprehensive and adaptable SDN solution. [59].

SD-WAN, originating from the SDN paradigm, utilises SDN mechanisms for operating, managing, automating, and simplifying networks within the WAN environment. Con-

sequently, SD-WAN has emerged as a promising solution to overcome the limitations of traditional WANs, with the objective of enhancing the performance and deployment of diverse data-driven applications [60]. The concept of separating the control and data layers, which is intrinsic to SDN, is extended to the SD-WAN ecosystem, along with the implementation of software-based centralised controllers. While SDN primarily oversees the control and administration of internal network functions within data centres (such as cloud and edge infrastructure), SD-WAN shifts its attention to managing the connections between geographically dispersed applications, systems, and data centres [61].

2.3.3 Bandwidth slicing

It is a strategy in network management that involves dividing the total network bandwidth into smaller segments, or "slices," to distribute among various users, applications, or services. This division is based on each entity's priorities, needs, or specific policies. The technique is especially beneficial in situations where equitable resource allocation, traffic prioritisation, or adherence to certain quality of service (QoS) standards are crucial.

By employing bandwidth slicing, networks can more effectively manage and allocate their resources, thereby boosting both performance and reliability. This method is particularly applicable in settings with a mix of applications requiring different levels of bandwidth, such as corporate networks, service provider infrastructures, or cloud-based platforms. Priority setting is a key feature of bandwidth slicing, where different bandwidth slices can be prioritised differently. This ensures that critical or urgent traffic is processed before less urgent or lower-priority data, optimising the network's overall efficiency and responsiveness [62].

2.3.4 Scheduling algorithms

Multi-level Queues is a sophisticated scheduling approach used in both operating systems and network management to efficiently handle tasks or data packets with diverse requirements. This method organises tasks or packets into multiple queues, each representing a different priority level or category of service. It's particularly useful in

environments where processes or packets have varied characteristics and requirements, allowing for more granular and differentiated handling based on predefined criteria [63].

First Come, First Served (FCFS) this scheduling algorithm is employed by operating systems and networks to oversee and execute tasks efficiently, processes, and requests in the order they are received. This method is also known by other names, such as first-in, first-out (FIFO) or first-come, first-choice (FCFC). The simplicity of the FCFS algorithm makes it highly predictable and capable of handling various types of tasks and requests without the need for complex prioritisation. It operates similarly to the queue system seen in grocery store checkouts, where customers are attended to in the order of their arrival, regardless of the transaction's complexity.

FCFS stands out as one of the most straightforward and self-sufficient scheduling algorithms, largely because it operates with minimal human or artificial intelligence (AI) intervention. It eliminates the need for task prioritisation based on urgency or complexity, thereby saving processing time. The central processing unit (CPU) itself undertakes the task of scheduling, obviating the need for external software or more sophisticated scheduling strategies [64].

Shortest Job Next (SJN), also known as Shortest Process Next (SPN) or Shortest Job First (SJF), this scheduling method priorities the process with the shortest execution time to run next. SJN operates as a non-preemptive algorithm, meaning it does not interrupt a process once it starts. A preemptive version of SJN is known as Shortest Remaining Time, which can interrupt processes to ensure the shortest one is being executed [65].

The primary advantage of Shortest Job Next lies in its straightforwardness and its effectiveness in decreasing the mean waiting duration for task execution. However, this method can lead to process starvation, where longer processes may end up waiting indefinitely if shorter processes keep coming in. A related strategy, Highest Response Ratio Next (HRRN), addresses this issue by implementing an ageing technique, which adjusts priority over time to prevent starvation [66].

2.3.5 Summary

In summary, traffic shaping is mainly concerned with managing data flow rates without priority consideration, and priority queuing, prioritises certain traffic, but it does not inherently guarantee resources for that traffic [67]. However, bandwidth slicing presents a comprehensive strategy that encompasses both prioritisation and the capacity for dynamic modification thanks to its ability to adapt and be customised and equity superior to that of conventional methods, ensuring that essential services have the bandwidth they need while avoiding resource monopolisation by any single user or application. For the scheduling methods, the multi-level queues method is flexible and capable of handling a diverse set of tasks based on specific needs and priorities, unlike FCFS, which processes tasks in sequential order, and SJN, which gives priority to shorter tasks irrespective of their urgency, and are less adaptable to varying task requirements and priorities [68].

In our efforts to improve the effectiveness and capabilities of the IEC continuum network, we have adopted an innovative approach that integrates bandwidth slicing techniques with a sophisticated multi-level queuing scheduling method. This method is meticulously implemented within the SDN controller, serving as a solution for optimising bandwidth allocation across the network and maximising the utilisation of network resources. More details have been discussed in Chapter 3.

2.4 Adaptive techniques for fault management and data quality in the IEC environment

Various detection methods have been employed, including statistical analysis, adaptive strategies, and machine learning. A thorough comparison of these three main approaches is discussed in [69]. In this study, our focus is on identifying and addressing faults as well as enhancing data quality through the use of machine learning. Machine learning has been extensively implemented across numerous research initiatives due to its simplicity, efficiency, and precision in managing faults and maintaining the integrity of data quality. Within machine learning, there are three main approaches: unsupervised learning, supervised learning, and reinforcement learning.

Supervised learning is a technique where a model is trained on labelled data, enabling the algorithm to understand the relationship between the inputs and their corresponding outputs. This approach is particularly effective for tasks such as classification and regression. One of the main benefits of supervised learning is its proficiency in making precise forecasts when there is a plentiful supply of labelled data. However, a notable drawback of this learning method is its reliance on having labelled data for training, which can make it impractical in situations where acquiring such data is costly or requires a lot of time [70].

Unsupervised learning operates with data that doesn't come with predefined labels, enabling algorithms to identify patterns or connections within the data themselves. This type of learning is useful in applications like clustering and reducing the dimensions of data. The strength of unsupervised learning lies in its ability to detect unseen patterns within datasets that lack labels. However, it encounters difficulties in assessing the value of the patterns it identifies, due to the absence of clear labels for verification[71].

Reinforcement learning is centred around an agent that engages with an environment, learning to make decisions in a sequence through feedback received as rewards or penalties. This approach is ideal for situations that demand independent decision-making (Figure 2.5). The effectiveness of reinforcement learning is particularly noted in environments where learning from direct experience is necessary, though it demands meticulous adjustments. Each methodology has its advantages and disadvantages, with the selection based on the particular issue at hand and the data accessible. In our thesis, we employed reinforcement learning techniques that are prominently recognised in academic literature for their effectiveness in dynamic and interactive settings.[72].

Here are the key components and concepts of reinforcement learning [73]:

Agent: This refers to the learner or decision-maker that operates within an environment, performing actions based on its interactions.

Environment: The context or the external system the agent interacts with, which reacts to the agent's actions by providing rewards or penalties as feedback.

State: A snapshot of the environment's current conditions, offering the agent infor-

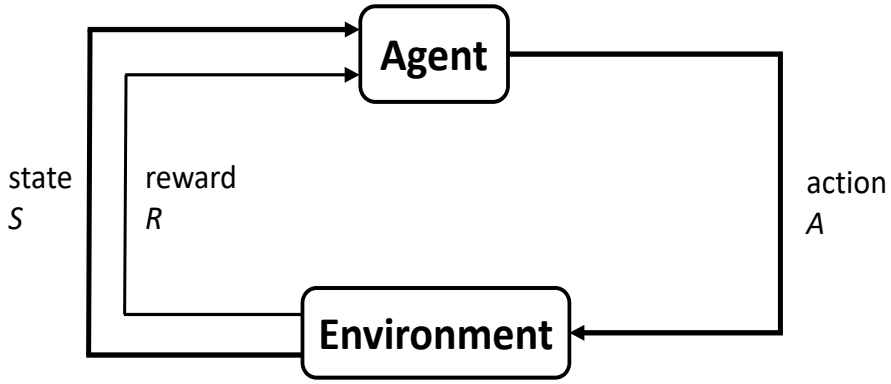


Figure 2.5: RL

mation to make informed decisions.

Action: The range of choices available to the agent in any given state, which can alter the environment's state.

Policy: The agent's strategy, is essentially a guide that maps states to actions, aiming to identify an optimal policy for maximising rewards over time.

Reward: The environment provides feedback to the agent's actions in the form of numerical values, representing rewards. The goal of the agent is to maximise the total rewards it receives over time.

Model-based: Involves the agent possessing a detailed model of the environment, including how actions transition between states and their rewards, to plan its actions.

Model-free: Here, the agent acquires optimal actions or value functions by directly engaging with the environment, devoid of a predefined model.

Value-based: Centres on learning a value function (like Q-values) that predicts The anticipated rewards for every pairing of state and action, with Q-learning as a prime example.

Policy-based: Focuses directly on learning the policy that dictates actions for given states, with methods like REINFORCE being key examples.

On-policy: Learning occurs based on the current policy in use, with SARSA being a notable on-policy learning algorithm.

Off-policy: The learning is based on the outcomes of actions taken under a different policy, with Q-learning exemplifying this approach.

Single-agent: Involves one agent engaging with the environment to learn the optimal policy or value function.

Multi-agent: Features several agents interacting among themselves and the environment in cooperative or competitive settings, each learning its policy.

Exploration: The method by which an agent investigates new actions or states to better understand the environment.

Exploitation: The agent's method for selecting actions that are known to maximise rewards according to its accumulated knowledge.

Continuous: Describes an action space with a continuous range, such as real numbers, with DDPG tailored for such spaces.

Discrete: Refers to an action space made up of a finite set of choices, with algorithms like Q-learning and SARSA designed for these spaces.

Batch Learning: The learning process from a pre-collected dataset of experiences.

Online Learning: The agent dynamically updates its policy or learning based on new experiences acquired from ongoing interactions with the environment.

2.4.1 Q-learning

Q-learning stands as a widely utilised form of RL, distinguished by its off-policy and model-free nature. Setting it apart from other reinforcement learning techniques, Q-learning is characterised by its straightforward Q-functions, making it a cornerstone upon which numerous other reinforcement learning algorithms are built [74].

It computes the execution value of a particular action within a given state. RL agents employ the Q-learning algorithm to acquire optimal strategies for managing faults and allocating resources within the IoT system. The Q-values (Q_r) are updated for each RL agent r $\in$ R using the Bellman equation as a basic concept. To iteratively update

the Q-values, it takes the present reward, the highest anticipated future reward, and the learning rate, according to the following equation:

$$Q_r(s,a) \leftarrow (1-\alpha) \cdot Q_r(s,a) + \alpha \cdot (r + \gamma \cdot \max a' Q_r(s',a')) \tag{2.1}$$

where:

- Q_r(s, a) refers to the Q-value of agent r in state s and performing action a.
- r refers to the immediate reward received from the environment.
- α refers to the learning rate, which controls the weight of the new information ($0 < \alpha \leq 1$).
- a' refers to the action taken in the next state in accordance with the existing policy.
- s' refers to the next state.
- γ refers to the discount factor, which specifies the significance of future rewards ($0 < \gamma \leq 1$).

Moreover, the balance between the weights assigned to the new information and the current information stored in the Q-values is determined by the α parameter. It regulates how frequently the Q-values are modified in response to the RL agent's new experiences gained. The γ parameter illustrates how future benefits are prioritised over present gains. It establishes how much consideration is given to potential benefits while making decisions. Therefore, we assumed the $\alpha = 0.1$ and the $\gamma = 0.9$.

2.4.2 State-Action-Reward-State-Action (SARSA)

SARSA is similar to Q-learning, but it operates as an on-policy algorithm, unlike Q-learning. This means SARSA updates its Q-values based on actions taken by the current policy, rather than choosing actions greedily based on the highest Q-value.

Unlike Q-learning, SARSA doesn't always use the maximum reward for updating its Q-values. Instead, it selects a new action and its corresponding reward using the same policy that guided the initial action selection. The acronym SARSA derives from the sequence used in the update formula: Q(s, a, r, s', a'), where 's' and 'a' represent the initial state and action, 'r' is the reward received afterwards, and 's', 'a' are the subsequent state and action pair. In contrast, Q-learning is not limited by the policy for choosing the next action; it selects whichever action maximises the Q-value for the following state. Hence, SARSA's characteristic feature is its adherence to the on-policy approach [75].

2.4.3 Deep Q-Networks (DQN)

The Deep Q-Network (DQN) It is a notable algorithm in the realm of RL, combining deep neural networks with the Q-learning method. This amalgamation enables agents to learn optimal strategies in complex environments. While Q-learning presents a robust framework, its critical limitation lies in its narrow applicability. Conceptualising Q-learning as a process of updating values within a two-dimensional grid (Action Space x State Space) reveals dynamic programming, highlighting a significant shortfall: the inability of Q-learning agents to make informed decisions regarding previously unencountered states. In essence, Q-learning lacks the foresight for valuing unknown states. DQN addresses this shortfall by replacing the conventional two-dimensional array with a neural network, using this network to approximate the Q-value function involves inputting the current state and generating estimated Q-values for all potential actions as output. Nonetheless, DQN's design is predominantly suited for offline batch learning, and it struggles with adapting to online, continuous learning contexts, where it's required to learn from a stream of incoming data continuously [76].

2.4.4 Deep Deterministic Policy Gradient (DDPG)

The DDPG algorithm is a model-free, online, off-policy method within RL. It utilises an actor-critic framework, where the agent, comprising both actor and critic components, endeavours to determine the best policy for maximising anticipated long-term rewards. While the DQN algorithm has seen significant success in complex, high-dimensional

environments like Atari games, it operates within a discrete action space. DDPG differentiates itself by utilising the actor to adjust the policy function's parameters, effectively choosing the best action for a given state, and the critic to assess the policy's performance based on the temporal difference (TD) error [77].

This method is particularly relevant for tasks requiring continuous action spaces, such as physical control tasks, where discrediting the action space too finely could result in an impractically large number of actions. For example, dividing the action space of a system with ten degrees of freedom into four parts each would result in over a million possible actions, making convergence challenging. Additionally, DDPG typically lacks in performing exploratory actions, which can be addressed by introducing noise into either the parameter or action space to encourage exploration. A comparison of the above reinforcement learning algorithms in Table 2.1

In conclusion, we choose Q-learning for our project, giving careful consideration to its unique benefits compared to other reinforcement learning (RL) algorithms. The off-policy characteristic of Q-learning is a key advantage, allowing the algorithm to learn from an exploratory policy. This is particularly valuable in settings where thorough exploration is crucial for identifying the best policies. In contrast to SARSA, which operates on-policy and might adopt a more cautious exploration strategy, Q-learning's off-policy nature facilitates quicker convergence by utilising experiences from various policies. Moreover, Q-learning is adept at managing infrequent rewards and efficiently navigating through complex state-action spaces. When compared to deep reinforcement learning algorithms like DQN and DDPG, Q-learning distinguishes itself with its straightforwardness and ease of use, establishing it as a strong option for a wide range of applications. Its appropriateness for scenarios with discrete actions and its efficient use of resources render it the optimal choice for our particular endeavour, especially in dynamic and challenging conditions.

Table 2.1: Reinforcement Learning Algorithms

Algorithm	**Type**	**Objective**	**Learning Approach**	**Action Space**	**Architecture**
Q-Learning	Model-free, Value-based	Learning the optimal action-value function $Q(s, a)$	Off-policy	Discrete	Not deep (traditional RL)
SARSA	Model-free, On-policy	Learning the optimal action-value function $Q(s, a)$	On-policy	Discrete	Not deep (traditional RL)
DQN	Model-free, Value-based, Deep Learning	Expand Q-learning to manage state spaces with a high dimensionality employing neural networks	Off-policy, Experience Replay	Discrete	Deep neural network (CNN)
DDPG	Model-free, Policy-based, Deep Learning	Learn a deterministic policy for continuous action spaces	Off-policy	Continuous	Actor-Critic (Deep neural networks)

2.5 Thesis scope in the context of adaptive time-critical data processing

2.5.1 Network optimisation in IEC

Various published works have extensively examined subjects relevant to the bandwidth dynamic management and resources in network systems, particularly for IoT applications. Diverse solutions and outcomes have been demonstrated on IoT devices, edge devices, and cloud servers.

bwSlicer [78] presents a system crafted to improve bandwidth allocation for VMs in

cloud data centres, with the goal of boosting performance and energy efficiency. It introduces three algorithms: Fair Bandwidth Reallocation (FBR), Required Bandwidth Allocation (RBA), and Divide Bandwidth Reallocation (DBR). Each algorithm tackles various facets of bandwidth management and allocation to cater to the changing requirements of VMs, consequently diminishing execution time and energy usage. These algorithms dynamically reallocate bandwidth among VMs based on their current demands and execution status, aiming for a more efficient utilisation of resources. This paper focuses on intra-datacenter bandwidth allocation among VMs without considering the broader network conditions and the potential for bandwidth optimisation between the data centre and other network segments or across multiple data centres. This thesis provides a comprehensive approach by extending the principles of dynamic allocation to include wider network conditions, potentially improving overall network efficiency and service quality beyond the scope of a single data centre.

In [62] the authors explore a distributed network slicing strategy for optimal allocation of both bandwidth and computational resources in 5G systems. It focuses on minimising latency by jointly allocating resources of base stations and fog nodes across a network, using a distributed optimisation algorithm. This approach aims to enhance service quality for end-users by efficiently managing communication and computational demands. This paper focuses on static, predefined resource allocation without real-time adaptability to fluctuating network conditions and demand. This thesis provides real-time bandwidth allocation adjustments based on immediate network performance and user demand, ensuring optimal resource utilisation and service quality across a broader range of network scenarios.

In [79] the authors delve into optimising data flow control within Time-Sensitive Networks (TSN) to ensure efficient load balancing across network paths in industrial automation settings. It addresses the unique challenges of achieving reliable and deterministic data transport in increasingly digitalis and connected industrial environments, influenced by the Industry 4.0 revolution. The work explores various load distribution strategies and the integration of TSN standards, emphasising dynamic control based on real-time traffic load metrics. It proposes solutions to minimise local load maxima and enhance network performance, considering the predictable nature of communi-

cation demands in automation applications. This paper focuses on optimising load distribution within a predefined network framework without dynamically adjusting network bandwidth allocations based on fluctuating demands and conditions. This thesis provides a complementary approach by offering real-time, adaptive bandwidth management, ensuring optimal network resource utilisation, thus potentially improving network performance and efficiency in dynamic industrial environments.

DART [80] presents a scheme to optimise traffic flow migration in Software-Defined Networks (SDN) by reducing data plane load. It introduces a method that ensures efficient migration of traffic flows without causing congestion or excessive rule-space usage in SDN switches. The key contributions include a QoS-aware migration schedule, feasible migration schedule generation, and rule-space management, aiming to minimise the migration impact on network performance. This approach facilitates smooth traffic flow migration, adhering to QoS demands and resource constraints. While DART aims to optimise flow migration to reduce load and avoid congestion, it does not directly manage the allocation of bandwidth across the network. This thesis provides a mechanism for real-time bandwidth management, ensuring optimal utilisation of network resources during flow migrations.

In [81] the authors delve into optimising video analytics at the edge by adaptively managing configuration settings and bandwidth allocation. It introduces an algorithm, JCAB, which dynamically balances the trade-offs among analytics accuracy, energy consumption, and system latency. While it shares the goal of efficient resource use with our work, our approach broadens the scope to include dynamic bandwidth slicing for a variety of IoT data streams, providing a more generalised solution that can adapt to diverse application requirements beyond video analytics.

JANUS [82] introduces a traffic scheduling system that prioritises data streams based on their latency requirements. The goal is to improve the performance of IoT applications within edge computing settings. It offers a nuanced approach to handling data streams with varying priorities, focusing on reducing latency and improving throughput. Janus dynamically manages traffic flow and bandwidth within the scope of edge networks by prioritising traffic based on latency requirements. However, this thesis contrasts Janus by managing and dynamically allocating bandwidth across the entire

network, i.e., IoT, edge, and cloud, offering a more holistic optimisation of network traffic and bandwidth utilisation across the entire network infrastructure.

In [83] the authors explore the utilisation of network slicing within vehicular networks to address diverse traffic demands and ensure efficient use of infrastructure for both safety and infotainment services. It models a highway scenario, employing a network-slicing approach to categorise vehicles into clusters for optimised communication. This method significantly enhances V2X communication by achieving low latency and high reliability, crucial for safety-critical applications in autonomous driving. This thesis extends beyond vehicular communication to encompass a variety of IoT scenarios, offering a more generalised framework for bandwidth management across the edge-cloud continuum. A limitation of this approach is the potential for static pre-configuration of slices, which might not optimally respond to the highly dynamic conditions of vehicular networks, such as varying speeds, densities, and the urban environment. This thesis offers more flexible and adaptive management of network resources, ensuring that the allocation of bandwidth can adjust in real-time, thus enhancing the network's responsiveness and efficiency.

NebulaStream (NES) [84] introduces a novel data management system designed for the IoT ecosystem. This system addresses the challenges of managing distributed, highly heterogeneous, and dynamic environments typical of IoT applications. NES emphasises efficient data and application management across a unified sensor-fog-cloud architecture, leveraging the strengths of edge computing to enhance scalability, reliability, and performance. It focuses on handling the unique characteristics of IoT data, such as its volume, velocity, and variety, by implementing dynamic decisions, autonomous processing, and incremental optimisations to manage data flows efficiently across the continuum. While NES efficiently manages data flow across the heterogeneous and unreliable IoT environment, it does not explicitly address dynamic network bandwidth allocation. This thesis offers adaptive network resource management, ensuring optimal data transmission rates across the IoT-edge-cloud continuum, further reducing latency and improving overall system performance under varying network conditions.

In [85] the authors present a novel data centre transport architecture, NDP (Network Data Plane), designed to address the challenges of achieving low latency and high

performance in modern data centre networks. This work introduces a system that sidesteps conventional transport protocol limitations by leveraging per-packet multi-path routing and innovative packet trimming techniques to manage network congestion and prioritise traffic effectively. The NDP project is centred on re-architectonic datacenter networks to optimise latency and throughput within a more confined network topology. While NDP efficiently manages data flow within datacenter networks, it doesn't dynamically adapt to changing network conditions or bandwidth demands outside of these environments. In contrast, this thesis's adaptability adjusts bandwidth allocations in real-time based on the varying requirements of different applications or services, enhancing performance in a wider range of network conditions and usage scenarios.

QJUMP [86] introduces a novel approach for controlling network interference in data centre networks, focusing on reducing queuing delays for latency-sensitive applications by allowing their traffic to "jump" over less sensitive traffic. This is achieved through a simple, deployable Linux Traffic Control module. QJUMP specifically targets data centre network interference with a focus on prioritisation and latency reduction. QJUMP addresses latency reduction through packet prioritisation within the more confined environment of data centre networks. QJUMP's reliance on static priority levels and rate limits, which might not adapt well to changing network conditions or varying application requirements. While QJUMP efficiently manages network queues to reduce latency for prioritised traffic, it does not dynamically adjust network bandwidth allocation based on real-time traffic patterns. This thesis provides a more flexible and adaptive mechanism for managing network resources, optimising not only latency but also bandwidth utilisation and overall network performance under diverse and fluctuating load conditions.

pHost [87] introduces a data centre transport design that combines the near-optimal performance of specialised transport designs like pFabric with the generality of using commodity network hardware akin to Fastpass. pHost achieves this by decentralising scheduling decisions to end-hosts, avoiding the need for specialised hardware or centralised scheduling. pHost focuses on optimising data centre transport using existing network infrastructure to achieve low flow completion times and high through-

put. While pHost effectively utilises existing network infrastructure to optimise data flow, it does not explicitly manage bandwidth allocation based on real-time network conditions or application requirements. This thesis contrasts pHost by dynamically adjusting network bandwidth allocation to meet the varying demands of IoT applications, potentially enhancing and maintaining performance under fluctuating network conditions and diverse application workloads.

LEO [88] introduces an advanced scheduling system, LEO, designed for mobile sensor applications. It efficiently manages sensor data processing across various computational units within smartphones, including CPUs, co-processors, GPUs, and cloud resources. By exploiting the specific characteristics of sensor app workloads, LEO significantly enhances energy efficiency and application performance without sacrificing accuracy or responsiveness. The LEO optimises computational resource allocation for mobile sensing applications. It focuses on computational offloading and scheduling without directly managing network bandwidth. While LEO optimises local and cloud processing to improve energy efficiency and latency, it may not fully account for network conditions that can vary widely and impact the performance of sensor-based applications. This thesis contrasts LEO by providing adaptive management of network resources, ensuring optimal data transmission rates, and further reducing latency and energy consumption in IoT sensing scenarios.

Homa [89] offers a novel transport protocol aimed at significantly lowering latency in data centre networks, particularly for workloads dominated by short messages. By dynamically managing in-network priority queues and employing a receiver-driven flow control mechanism, Homa achieves exceptionally low latencies and high network utilisation. Homa provides a robust solution for maintaining low latency across a wide range of network loads and message sizes. Homa's focus is on optimising transport protocol behaviour within the constraints of existing network infrastructure. While Homa effectively manages packet priorities and receiver-driven flow control to reduce latency, it does not directly address the dynamic allocation and optimisation of network bandwidth based on varying application demands and network conditions. This thesis contrasts Homa by providing a network layer solution to adaptively manage bandwidth allocation in real-time, ensuring optimal utilisation of network resources.

Frontier [90] explores a distributed, data-parallel edge processing platform designed for IoT applications. By utilising replicated dataflow graphs (RDGs) and backpressure stream routing, Frontier aims to enhance throughput and resilience in the face of intermittent network connectivity and dynamic wireless network conditions. This method utilises varied network paths and selective network-aware replay mechanisms to dynamically direct data, ensuring efficient processing even with varying network bandwidth. Frontier focuses on distributed data processing at the edge, enhancing resilience and adaptability to network changes. Frontier's emphasis on leveraging multiple edge devices for parallel data processing and its novel strategies for overcoming network volatility. Frontier's reliance on network path diversity and local processing strategies without explicitly managing network bandwidth. While Frontier efficiently utilises the available network and computational resources of IoT devices, it may not fully address scenarios where bandwidth constraints are a significant bottleneck. This thesis provides a mechanism to dynamically allocate and optimise network bandwidth, ensuring data streams are not only processed efficiently but also transmitted optimally across the network.

ApproxIoT [91] the emphasis is on enhancing the effectiveness of data analytics in IoT environments via approximate computing. This method involves processing a representative subset instead of the complete dataset, providing a structured balance between result precision and computational effectiveness. It employs an online hierarchical stratified reservoir sampling technique to generate approximate results with well-defined error margins, prioritising speed and resource utilisation. ApproxIoT focuses on reducing computational demands by processing subsets of data, which may lead to scenarios where data transmission delays impact the timely analytics and decision-making at the edge. This thesis with its ability to dynamically adjust network resources based on priority and utilisation, could complement ApproxIoT by ensuring that data packets critical for analytics are prioritised and transmitted efficiently, thus mitigating potential network-related delays or congestion that could affect the analytics' accuracy and timeliness.

In [92] the author introduces a Software-Defined Networking (SDN)-based queuing framework, SDQ, aimed at enhancing Quality of Service (QoS) in 5G networks. By

leveraging SDN's capabilities, SDQ dynamically selects optimal queues and paths for incoming flows to manage workload imbalances and ensure low latency. This approach represents a significant advancement in traffic engineering for networks, focusing on deterministic QoS support. The former concentrates on the application of SDN for queue and path optimisation in 5G networks. While SDQ focuses on improving QoS within the constraints of 5G networks, this thesis offers a more adaptable solution across various network types and conditions, especially in heterogeneous environments typical of IoT ecosystems. This adaptability could enhance QoS provisioning for a wider range of applications, addressing limitations in fixed infrastructure and specific network technologies.

2.5.2 IEC devices sustainability

Numerous published papers have thoroughly analysed topics related to energy optimisation and resource-constrained environments, especially for IoT applications. Various solutions and results have been shown on IoT devices, edge devices, and cloud servers.

In [93] the author proposes a novel cross-layer approach to optimise energy consumption and avoid routing collisions in Wireless Sensor Networks (WSNs). This strategy integrates information from the physical and MAC layers to make routing decisions, aiming to extend network lifespan and improve data transmission quality. The method involves using Link Diagnostic Values (LDV) and Energy Identification (EI) for routing decisions, emphasising the efficiency of multi-hop communication paths. This paper primarily focuses on network-level optimisations to reduce energy consumption and avoid data collisions. This thesis, in contrast, delves into device-level optimisation using reinforcement learning to dynamically adjust data transmission rates based on energy availability, offering a more granular and adaptive approach to energy optimisation.

In [94] the author utilises RL-driven adaptive fuzzy logic systems and genetic algorithms to enhance energy efficiency in IoT devices. This approach focuses on balancing energy usage and performance parameters, ensuring efficient operation and minimising energy consumption. The core mechanism involves an Intelligent Agent that adapts device behaviour based on environmental conditions to achieve energy conservation.

This system represents a comprehensive AI-based architecture for efficient energy management in IoT devices, aiming to enhance system performance and security without compromising on energy efficiency. While both papers utilise RL, this thesis offers a solution to the limitation of scalable and adaptive energy management in time-critical situations, directly addressing the challenges of maintaining device operation in varying environmental conditions and energy availability scenarios not covered by this paper. It thus provides a more dynamic and context-aware energy management strategy for IoT devices.

In [95] the author concentrates on a model for a green energy system aimed at improving energy efficiency in smart cities across various domains such as street lighting, buildings, signage, residential properties, and parking facilities. It utilises IoT-based sensors and controllers to manage energy consumption dynamically, ensuring that devices operate efficiently and only when needed. Unlike the static IoT-based control systems described, this thesis employs reinforcement learning, offering a more adaptable approach to energy optimisation that can dynamically respond to changing environmental conditions and device states. Also, it can cover limitations related to the flexibility of energy management strategies in the face of unpredictable energy availability, especially in scenarios not covered by traditional IoT frameworks.

In [96] the author introduces a hybrid model combining the Whale Optimisation Algorithm (WOA) and Simulated Annealing (SA) to optimise Cluster Head (CH) to optimise the energy consumption of wireless sensor network (WSN) sensors in IoT systems. The objective is to prolong the lifespan of the IoT system and improve network efficiency. The paper focuses on a specific metaheuristic combination for energy optimisation, which might not dynamically adapt to changing network conditions or device states as effectively as reinforcement learning approaches. This thesis, leveraging reinforcement learning, offers a more dynamic and adaptive solution capable of real-time adjustments to energy consumption and operational parameters, potentially covering limitations related to fixed algorithmic performance in varying conditions.

Q-EBIoT [97] it presents a quantum-inspired green computing framework aimed at bolstering energy efficiency within IoT networks. This innovative approach introduces energy-centric Q-bit representation and optimisation methods to prolong network lifes-

pan and enhance the effectiveness of sensor networks. This paper leverages quantum computing concepts to address energy optimisation, offering a different methodological perspective on achieving energy efficiency in IoT systems. The quantum framework proposed is reliant on advancements in quantum computing, which may limit its immediate applicability in current IoT systems. This thesis, on the other hand, is grounded in current RL capabilities, making it more immediately deployable in existing IoT infrastructures. It allows for dynamic adjustments based on energy availability and operational context, potentially offering more granular energy optimisation compared to the broader, quantum-based strategy outlined in the paper. This could result in more efficient energy use in real-world scenarios, where conditions change rapidly and unpredictably.

In [98] The author suggests an inventive energy management model to enhance the efficiency of data dissemination in Wireless Sensor Networks (WSNs) within IoT systems. The model aims to optimise energy consumption during data transmission, thus prolonging the network's lifespan and enhancing its overall efficiency. The model in the paper might be less adaptable to changing environmental conditions or operational requirements due to its focus on pre-determined optimisation paths. While this thesis leveraging reinforcement learning, inherently adjusts to new information and conditions, potentially offering a more flexible solution for energy optimisation in diverse IoT environments.

In [99] the author investigates energy efficiency in IoT systems via an energy harvesting protocol. The emphasis is on crafting a framework that optimises energy usage among interconnected devices within intricate and time-sensitive IoT setups. This approach is aimed at enhancing the sustainability and lifetime of sensor networks within IoT environments by addressing challenges in data delivery, quality of service optimisation, and design of MAC protocols that consider energy consumption beyond data transmissions. This research presents a distinct methodology concentrating on protocol design and energy harvesting to improve network efficiency and device longevity. The paper's approach is limited in adaptability to real-time changes and diverse IoT applications. This thesis can extend its impact by applying adaptive learning to a wider range of IoT scenarios, ensuring optimal energy use across various operational contexts. The rein-

forcement learning approach could enhance the paper's proposed energy optimisation strategies by integrating predictive analytics and adaptive decision-making, offering improvements in energy efficiency and network performance.

In [100] the author introduces an IoT-centred system engineered to manage energy intelligently in buildings. They propose a semantic framework for standardised modelling and a web-based tool for managing real-time energy data. This approach aims at optimising energy consumption through actionable insights derived from real-time and predicted data. This paper focuses on a specific application area of intelligent building management, employing IoT technologies for data integration and analysis to facilitate energy savings within the built environment. One of the limitations is the system's lack of capabilities for processing real-time data and automatically implementing action plans without human intervention for more dynamic and efficient energy management. This thesis, leveraging reinforcement learning, directly addresses this limitation by providing a dynamic, adaptive framework for energy optimisation in IoT devices. It offers real-time adaptability to changing energy availability and operational demands, ensuring energy efficiency without compromising the system's responsiveness or performance.

In [101] the author introduces an energy-efficient protocol using fuzzy logic and an immune-inspired algorithm for cluster head selection and routing in IoT networks. It aims to enhance network lifetime and data delivery reliability by optimally selecting cluster heads and routing paths, Integrating the Fuzzy Analytic Hierarchy Process (FAHP) combined with the Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS) for decision-making. This paper's limitation lies in its specific focus on clustering and routing within WSNs. Comparatively, this thesis offers broader applicability and adaptability across various IoT scenarios, not limited to WSNs, by dynamically adjusting device operations in real-time based on energy availability and operational demands.

In [102] the author outlines a distributed IoT platform designed to enhance energy management across urban districts. It combines information sourced from System Information Models (SIMs), Building Information Models (BIMs), and Geographic Information Systems (GIS) with real-time data from IoT devices for thorough en-

ergy analysis and simulation. This approach aims to optimise energy distribution and consumption, focusing on smart city services. This paper develops a platform for integrating various data sources to manage and simulate energy in urban settings. One limitation of the presented platform might be its focus on simulation and policy evaluation without explicitly detailing adaptive, real-time operational adjustments at the device level, which is a gap this thesis addresses through RL-driven dynamic management for energy optimisation.

In [103] The author proposes an offloading method using a Self-Adaptive Particle Swarm Optimisation Algorithm combined with Genetic Algorithm operators (SPSO-GA) to improve energy efficiency in cloud-edge computing environments. This strategy aims to reduce energy consumption by efficiently offloading deep neural network (DNN) layers between edge, cloud, and IoT devices, considering the computational constraints and energy limitations of IoT devices. This paper presents a method to manage computational offloading across cloud-edge architectures to minimise energy usage. A significant limitation of this paper could be its focus on offloading strategies without explicitly addressing the dynamic adaptability of these strategies to varying operational contexts and energy conditions in real-time. This thesis, leveraging reinforcement learning, directly addresses this limitation by providing a dynamic, adaptive framework for energy optimisation in IoT devices. It offers real-time adaptability to changing energy availability and operational demands, ensuring energy efficiency without compromising the system's responsiveness or performance.

In [104] the author proposes a hierarchical structure to enhance the energy efficiency of IoT systems, allowing sensors to enter sleep mode based on predefined conditions, such as a low battery or when sensing is not required, aiming to extend resource lifespan and improve system operation. While the paper provides a structured approach to managing sensor states for energy savings, it may lack the flexibility and real-time adaptability of reinforcement learning techniques. This thesis potentially addresses these limitations by employing RL to continuously learn and optimise device operations, offering improvements in energy efficiency that can dynamically adjust to varying conditions and requirements in IoT systems.

In [105] the author proposes a cross-layer-based energy optimisation algorithm (CEOA)

for IoT systems. It aims to improve energy efficiency by integrating AI techniques with IoT, focusing on massive data management and device operations optimisation. This paper introduces a strategic framework combining AI and IoT at a systemic level, emphasising cross-layer optimisation to enhance energy efficiency across IoT networks. However, one limitation of this paper includes adaptability to rapidly changing network conditions This thesis with dynamic reinforcement learning addresses such a potential limitation by offering a flexible and adaptive approach to energy optimisation capable of real-time adjustments to diverse and unpredictable IoT environments.

2.5.3 Data quality monitoring and healing in IEC

Many scholarly articles have extensively examined subjects of data quality monitoring and healing, particularly in the context of IoT applications. Diverse solutions and outcomes have been shown on IoT devices, edge devices, and cloud servers.

In [106] the author introduces a novel architecture that integrates fault detection, recovery mechanisms, and deployment automation across cloud and edge resources. The framework focuses on optimising resource usage and reducing failure impact through proactive and reactive measures, ensuring high availability and resilience of IoT services. It employs a combination of offline optimisation and online adaptation to manage application deployment and fault recovery, leveraging cloud resources as backup to meet end-to-end latency requirements. This thesis offers the ability to learn and adapt from interactions within the IoT environment, which provides a dynamic and efficient way to handle evolving system configurations and emerging faults, which may not be explicitly addressed in the fault-tolerant workflow composition framework. Also, emphasis on agent collaboration offers enhanced resilience and efficiency through distributed problem-solving, which might not be as central in the multi-cloud edge framework's approach.

In [107] the author explains a technique for detecting high-impedance faults using transfer learning within a collaborative framework between cloud and edge computing. It introduces a system utilising power distribution IoT, divided into terminals, edges, and cloud layers. This method focuses on efficiently detecting high impedance faults (HIFs) by utilizing information from various distribution networks and employ-

ing convolutional neural networks (CNNs) for feature extraction and fault detection. The system is designed to work in both online and offline modes, allowing for real-time detection and model updating without the need for continuous data upload. A potential limitation of this approach is its adaptability and scalability to a wider range of IoT applications beyond high-impedance fault detection in power distribution. This thesis offers more flexible and comprehensive solutions for fault detection and system optimisation across various IoT scenarios, providing broader applicability and enhancing system resilience more generally.

In [108] the author outlines an innovative approach focusing on fault tolerance within IoT applications, specifically tailored for edge computing contexts. It emphasises the balance between latency awareness and adaptive fault tolerance, leveraging checkpointing and replication techniques to ensure the reliability and efficient execution of IoT applications. The strategy is designed to minimise latency and network traffic, effectively utilising available edge resources and improving system resilience against node failures. A potential limitation of the adaptive fault-tolerant strategy might be its reliance on the availability of idle edge nodes and the efficiency of checkpointing and replication processes, which could be challenged by highly dynamic or resource-constrained environments. This thesis covers this limitation by leveraging the dynamic decision-making capability of MARL to adaptively respond to changing conditions without predefined replication or checkpointing strategies.

In [109] the author introduces a comprehensive approach to enhancing IoT system resilience through self-healing mechanisms. It outlines a series of patterns or best practices designed to detect errors and recover from failures automatically, aiming to minimise system downtime and maintain service availability. The patterns are divided into two primary groups: error detection (probes) and restoration and upkeep of health, encompassing diverse tactics ranging from action audits to redundancy and dynamic adjustments for fault resilience. While the pattern language aims at general applicability across IoT systems, offering straightforward implementation guidance, it may lack the depth in handling complex, dynamic fault scenarios that this thesis excels in, through its ability to learn and adapt from environmental feedback and agent interactions.

In [110]The author explores an automated failure recovery framework designed for container-based IoT edge applications, with an emphasis on minimising downtime by swiftly identifying and restoring operations. It uses container deployment techniques for efficient management and recovery from failures in IoT applications deployed on edge nodes. While the paper may effectively address automatic failure recovery at a technical and operational level, it might not cover the broader aspects of system configuration and adaptation to prevent such failures. This thesis with MARL could complement this by providing a proactive approach to detect and rectify potential misconfigurations before they lead to failures, thus covering a limitation potentially not addressed by the paper.

In [111] the author introduces an innovative Fast Fault Detection Manager (FFDM), strategically enhancing fault detection and recovery in IoT services deployed in cloud environments. This work meticulously integrates VM and container orchestration systems, notably OpenStack and Kubernetes, to significantly streamline fault detection and recovery processes, aiming for minimal service disruption. The paper is distinct in its approach, focusing on infrastructure and deployment optimisation to accelerate fault management, a critical aspect for maintaining high availability in cloud-based IoT applications. However, the paper's limitation lies in its focus on the infrastructure and deployment level, primarily enhancing the mechanical aspects of fault detection and recovery without deeply integrating adaptive learning mechanisms or considering the dynamic nature of IoT environments and the potential for misconfiguration-related issues. In contrast, this thesis leveraging Multi-Agent Reinforcement Learning (MARL) for optimising IoT systems addresses these limitations by introducing a method that not only detects and handles faults but also learns from them to prevent future occurrences.

IoTEF [112] presents a novel architecture designed to enhance fault tolerance in IoT applications through a federated management system that spans cloud and edge computing environments. It emphasises the use of container-based virtualisation and microservices to facilitate seamless deployment and management across diverse computing resources. This architecture is particularly focused on achieving fault tolerance through data replication, exactly-once data semantics, and a unified management in-

terface for orchestrating heterogeneous clusters. A potential limitation of IoTEF that this thesis approach could address is the adaptability in rapidly changing environments and learning from system interactions to improve fault management strategies dynamically. While IoTEF offers a comprehensive framework for fault tolerance through architectural means, this thesis approach could complement it by providing adaptive, learning-based solutions to enhance system resilience further.

In [113] the author delves into the creation of a machine learning (ML) framework aimed at the proactive monitoring and prediction of the ageing process in IoT devices. Utilising cost-effective embedded tags, this framework facilitates operability across both edge and cloud computing environments. The methodology encompasses calculating device ageing factors and predicting future trends with optimised ML algorithms, ensuring accurate lifetime predictions. This process involves intricate steps like anomaly detection, utilising TensorFlow for deep neural network (DNN) training, and implementing edge ML to balance accuracy and privacy. Comparatively, this thesis addresses a set of challenges, focusing on system-wide misconfiguration and fault management rather than the individual device's physical ageing process. The MARL-based framework might extend the scope of the paper by offering a dynamic system-level management solution that complements the device-level ageing predictions. It covers the gap in dynamic adaptation to changing operational conditions and configurations, potentially enhancing the overall resilience of IoT systems against both physical ageing and configuration-related failures.

In [114] the author introduces an advanced system for optimising fault recovery in IoT environments, utilising a self-healing mechanism that leverages efficient failure detection (PE-FD) and energy-optimal task assignment. This system significantly enhances the resilience and energy efficiency of IoT applications by dynamically reallocating tasks across nodes in response to failures, ensuring minimal energy consumption and operational disruption. The paper showcases a methodical approach to failure detection, employing accrual failure detection augmented with adaptable policies for varying application needs and an Integer Linear Programming (ILP) model for task reallocation to achieve optimal energy usage. One limitation of the self-healing approach could be its reliance on predefined optimisation models and failure detection

policies, which may not adapt in real-time to unforeseen system dynamics or learn from past interactions as effectively as MARL-based methods. In contrast, the MARL approach covers this limitation by continuously learning from the environment and agent interactions, potentially offering more flexible and adaptive solutions to fault management and system optimisation in rapidly changing IoT scenarios.

SEDGE [115] introduces an advanced IoT system that efficiently adapts to new configurations and protocols while focusing on energy efficiency. It highlights novel contributions such as easy configurability, constant performance monitoring with self-healing actions, and edge AI-based services for local data processing. It focuses on interoperability and efficient service delivery at the edge, leveraging AI for real-time processing. However, while this paper provides a structured approach to self-healing and interoperability, it may not fully explore the potential of adaptive, agent-based learning for continuous system improvement and optimisation across varying conditions and configurations. In contrast, this thesis's MARL-based optimisation work might cover the dynamic management of system configurations and faults through learning-based strategies, thereby offering a more flexible and comprehensive solution to system optimisation and fault management in IoT systems.

In [116] the author presents a sophisticated model for diagnosing and visualising faults in high-speed trains, leveraging edge and cloud computing alongside a deep neural network (DNN) featuring Stacked Auto-Encoders (SAES-DNN). This innovative approach enhances fault diagnosis by integrating visual analysis through knowledge graphs, significantly improving the accuracy and speed of fault identification compared to traditional methods. The model effectively utilises cloud computing for heavy computational tasks and edge computing for real-time data processing. Contrasting with this thesis, the train fault diagnosis study focuses on specific fault detection and visualisation in high-speed trains. This thesis explores adaptive system management across diverse IoT environments, leveraging the flexibility of MARL for broad applications.

In [117] the author elaborates on enhancing IoT fault detection through a novel integration of edge computing, blockchain technology, and an improved Random Forest (RF) algorithm, dubbed the Data Set Accuracy Weighted Random Forest (DAWRF). This approach is refined further by Particle Swarm Optimisation (PSO) to fine-tune the

algorithm's parameters, aiming at higher fault detection accuracy and reliability. The use of blockchain ensures data integrity and accuracy verification, while the DAWRF algorithm leverages weighted voting based on out-of-pocket data and predictive test data sets to improve fault detection performance.

One limitation of this paper is the static nature of the proposed DAWRF algorithm's parameters and its focus on a singular method of fault detection. While the DAWRF algorithm provides an innovative solution for fault detection with improved accuracy, it does not inherently adapt to evolving system dynamics or learn from new data patterns over time. This thesis with MARL introduces a dynamic, adaptive approach that continuously learns from the environment to optimise system performance and fault management strategies. This learning-based approach not only covers the static detection capabilities but also extends to proactively adjusting system parameters and strategies in response to changing conditions, thereby offering a more flexible and comprehensive solution for IoT system resilience.

In [118] the author proposes a novel approach to task allocation and fault tolerance in IoT applications through decentralised edge computing. The methodology emphasises latency-aware distribution of tasks across a federation of edge nodes, enhancing performance and reliability without the need for centralised control. It presents a new algorithm designed to form groups and allocate tasks among edge nodes, considering their resource availability and latency needs, ensuring efficient execution of tasks within their deadlines. One limitation of the decentralised latency-aware approach is that it does not cover the dynamic adaptability in highly volatile environments where system configurations and operational conditions frequently change. While the decentralised approach efficiently handles task allocation and fault tolerance within a given framework, this thesis can provide a more flexible adaptation mechanism by learning from and evolving with the system's operational dynamics, potentially offering enhanced optimisation in scenarios where rapid changes in the environment or task requirements occur.

In [119] the author presents an innovative controller designed to enhance security and reliability in IoT and containerised environments. Utilising Hierarchical Hidden Markov Models (HHMMs) and Markov Decision Processes (MDPs), the controller

adeptly detects, identifies, and recovers from misconfigurations. While this work provides a structured, model-based approach for managing specific misconfiguration challenges, one limitation of the self-configuration controller might be its reliance on predefined models, which could limit its adaptability to unforeseen misconfigurations or novel system dynamics. In contrast, this thesis approach inherently adapts to new challenges through continuous learning, potentially offering a more versatile solution to the evolving landscape of IoT system management.

In [120] the author introduces an extensive cloud-based framework for IoT, specifically tailored for detecting and pinpointing faults in power distribution systems with high efficiency. Making use of sparsely positioned current sensing devices (CSDs), the system employs a context-aware strategy at the edge device (ED) level to smartly filter and preprocess measurements. This ensures that only relevant data indicating significant changes are transmitted to the cloud, reducing unnecessary data transfer and enhancing fault detection accuracy. While the cloud-based IoT solution provides a targeted approach for fault detection in power systems, emphasising data efficiency and precise localisation, it may face limitations in adaptability and real-time learning in diverse IoT environments. This thesis MARL-based approach covers these limitations by enabling systems to learn from interactions and adapt to new challenges dynamically, offering a more flexible solution for IoT system optimisation across various applications.

In [121] the author thoroughly investigates the creation of an AI-driven Intrusion Detection System (IDS) tailored for IoT networks, particularly emphasising the Routing Protocol for Low-Power and Lossy Networks (RPL). The approach involves generating large datasets through simulations to cover various attack scenarios on RPL-based networks. Utilising Machine Learning (ML) techniques, the system is trained to detect and classify different types of routing attacks effectively. The paper details the process of feature selection, dataset generation, and the evaluation of different classifiers, ultimately implementing a Random Forest (RF) classifier due to its high accuracy, computational efficiency, and feature selection capability. While the IDS paper focuses on security aspects, specifically intrusion detection within IoT networks using ML techniques, a limitation of the IDS approach is its focus on predefined threat mod-

els and reliance on static datasets for training, which may not fully account for novel or evolving attack vectors. In contrast, this thesis's MARL-based approach dynamically adapts to new challenges and system conditions, potentially covering the adaptability gap in static IDS models by learning from ongoing system interactions and adjusting strategies in real time to enhance system resilience against a wider range of issues, including those not initially anticipated.

In [122] the author explores an advanced fault diagnosis strategy for IIoT systems, utilising edge computing and the Random Forest algorithm to enhance diagnostic accuracy and efficiency. This method optimises the deployment of Programmable Logic Controllers (PLCs) at the edge of the network, enabling rapid and reliable fault detection within industrial settings. While this paper provides a precise, algorithm-driven solution for fault diagnosis in industrial environments, focusing on minimising hardware resources and computational overhead, it may not fully address the dynamic adaptability and scalability challenges inherent in diverse IoT applications. This thesis MARL-based approach seeks to cover these limitations by offering a flexible framework for continuous learning and adaptation, ensuring system robustness and efficiency in the face of evolving operational conditions and unforeseen system dynamics.

A summary table is provided, compiling the literature review alongside the current challenges 2.2.

Table 2.2: An overview of the literature review, including the primary challenges tackled in this thesis, is presented

Problem	**Related Works**	**Challenges**
Network optimisation in IEC	[78], [62], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92]	• Some frameworks are specific only for VMs on the cloud. • Most frameworks are static and not adaptable to changes in network conditions. • Most frameworks do not manage bandwidth allocation across the network i.e. IoT, edge, and cloud networks. • Some frameworks manage data flow only not considering network bandwidth allocation.
IEC devices sustainability	[93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [105]	• Some frameworks focus on the network level, not the device level. • Some frameworks do not consider time-critical applications. • Some frameworks use static strategies not adaptive strategies. • Some frameworks are only focused on IoT not considering edge and cloud.
Data quality monitoring and healing in IEC	[106], [107], [108], [109], [110], [111], [112], [113], [114], [115], [116], [117], [118], [119], [120], [121], [122]	• Most frameworks do not manage data quality and healing across the IoT and edge environment. • Several frameworks rely on predefined static optimisation models and failure detection policies, lacking real-time adaptation to environmental changes. • Most frameworks do not specify or identify the faults that happen in the system.

3

Dynamic Bandwidth Slicing for Time-Critical IoT Data Streams in the IEC

Contents

Summary

In recent years, edge computing has emerged as an important complement to cloud computing, especially for applications that demand Time-Critical communication guarantees, such as industrial control systems. Although edge computing enables real-time analysis of data streaming from IoT devices, these devices often don't have sufficient computing power to ensure effective performance for Time-Critical applications. A common solution to this challenge has been to transfer the data analytics tasks from edge devices to the cloud. However, this approach is typically static and does not adjust to changes in workload and network conditions. To address these challenges, we introduce an innovative distributed system with a Quality of Service (QoS)-oriented, multi-level queue traffic scheduling mechanism that supports semi-automatic division of bandwidth. This system is designed to efficiently manage urgent incoming traffic in edge-cloud settings, resulting in significant improvements in latency and throughput, along with a reduction in energy consumption for these environments.

3.1 Introduction

IoT is an emerging paradigm that shifts routine daily workloads into smart, automated mechanisms by gathering and processing an unprecedented amount of data in a continuous manner [123]. It tracks and monitors surrounding activities (e.g., automated industrial setup) to make better decisions, increase efficiency, and improve the quality of life. Coinciding with this paradigm, IoT-based applications adopt several integrated ecosystems – from edge and cloud computing to SDN and SD-WAN [9, 10]. Each ecosystem offers rich features to process and transmit data according to the given QoS of IoT applications.

The IoT paradigm with its associated industrial ecosystems delivers unprecedented advances in technological developments. However, its heterogeneous computing and network elements still encounter two fundamental problems, which can be defined as (1) a transmission mismatch and (2) a processing mismatch [124, 125]. The former problem occurs when incoming data streams at a given network arrive faster than the network can handle and transmit. This is typically due to several reasons, such as

the spike and fluctuation of incoming data and the instability of network connectivity between IoT ecosystem elements (senders and receivers)[126], [127], [128]. On the other hand, the processing mismatch problem arises when a given computing resource cannot process its incoming requests immediately or in a timely fashion due to the sharing mechanisms of computing resources [129, 130]. These two problems must be dealt with, especially in the context of real-time IoT applications where network and/or processing delays could lead to catastrophic incidents.

The two problems mentioned above have been tackled in different ways. For example, a data buffer technique is one typical solution that holds new arrival data for a period of time before being processed [131, 132]. Another typical solution is the leverage of classical congestion control mechanisms where new incoming data are dropped when a given buffer is overloaded [133]. Such techniques suffer from non-negligible delays at both transmission and processing levels, especially when IoT applications are latency-sensitive. Also, dropping any part of data introduces a further problem that would lead to data inconsistency with serious consequences in domains such as Industrial IoT [134]. Moreover, such techniques ignore the power of priority mechanisms at both network and host levels, which can hardly guarantee the quality of QoS for Time-Critical IoT applications.

Several efforts have been made to address the problem of transmission and processing mismatching. For example, [86] leverage computation offloading mechanisms where data and tasks that require intensive computational resources are forwarded to an external platform (e.g., cloud data centres). Another study [135] explores a congestion control approach focusing on tuning data transmission rates based on QoS requirements. However, the usefulness of these studies is limited to conventional environments (e.g., cloud data centres, edge computing) without considering the bigger range of IoT ecosystems along with cutting-edge approaches, such as dynamic network slicing, load-balancing, and prioritisation.

Overall, this chapter tries to solve the research question:

What is the best way to satisfy the latency constraints along with accelerating data transmissions for IoT Safety-Critical applications?

To address the research question this chapter presents a novel distributed IoT framework which is based on a multi-level network-host queuing mechanisms, prioritisation, and SDN network traffic slicing. The system is designed to make the best utilization of network and host resources in the edge-cloud Continuum (see Figure 3.1). It diminishes queuing delays and increases the QoS assurance of IoT applications with high-latency sensitivity as much as feasible. Given that the solutions presented are for time-critical applications, the QoS parameter of time takes precedence over other parameters. Consequently, trade-offs are consistently made to ensure minimised latency, prioritising rapid response times above other considerations.

To do so, our proposed system deploys global network agents in SDN and SD-WAN controllers for data stream scheduling based on prioritisation along with slicing bandwidth based on each IoT stream priority. The system also deploys IoT agents within each node (e.g. edge nodes, cloud nodes) to schedule IoT task executions based on multi-level queuing and prioritisation. Given these systems, we formulate two different optimisation problems to find the best solution for every IoT application such that the overall execution time is minimised while network bandwidth is utilised at maximum.

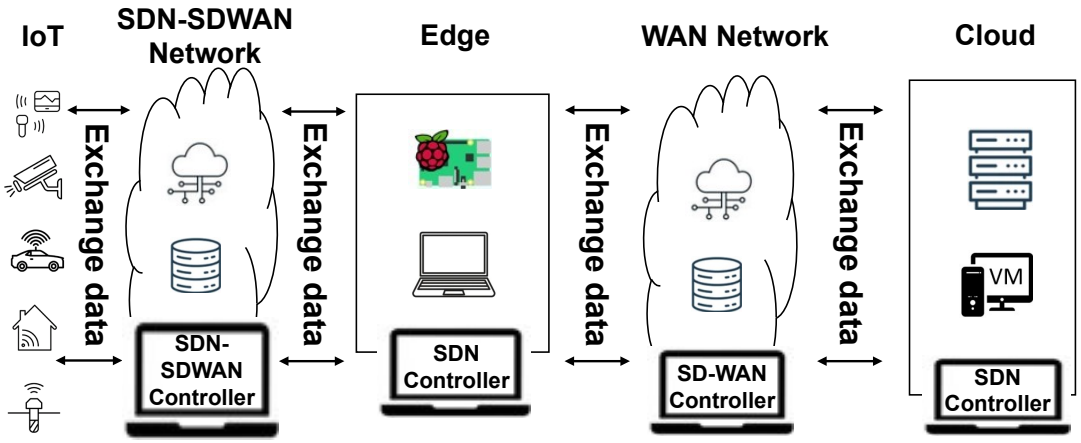


Figure 3.1: IoT-edge-cloud continuum modular architecture

Solving the above question might lead to insufficient use of network resources. This can be formalized in a question context as "**How can we indicate the network slicing percentage among several priority lists such that every slice is fully used by every list?**". It is known that network bandwidth is a scarce resource where network slicing percentage should be divided according to application priority ranks. One simple solution is to use a static percentage value for each list (e.g., 50%, 30%, and 20% for

three lists of high, medium, and low respectively). However, sometimes a network bandwidth slice is not fully used by a given priory list, which leads to insufficient use of network resources. To solve this problem, we propose a heuristic auto-adaptation algorithm to dynamically tune bandwidth slicing depending on the observed network utilisation of every priority.

In summary, the contributions of this chapter are as follows:

- we formulate the transmission and processing mismatch problem in the edge-cloud environment,
- we propose a novel distributed and QoS-based multi-level queues traffic scheduling system,
- we evaluate the performance of our proposed approach using a self-driving car test case scenario.

3.2 Formal model

In this section, we present the system description necessary to represent our research problem (Section 3.2.1). Using these definitions, we formulate our problem (Section 3.2.2). Table 5.3 summarizes all notations used in the chapter.

3.2.1 System overview

Our infrastructure system X consists of four infrastructure elements and is represented as a quadruple $\langle \mathcal{D}, E, C, N \rangle$. $\mathcal{D}$ is a set of IoT devices $\mathcal{D}_i$ and is denoted by $\mathcal{D}_i = \{id_i, \delta_i\}$. Here, id_i represents the identifier of the IoT device $\mathcal{D}_i$ and δ_i represents the data rate of IoT device $\mathcal{D}_i$. E is a set of edge devices E_e with each $E_e = \{id_e, h_e\}$. id_e and h_e represents the identifier and the set of host machines $h_{e1}, h_{e2}, ...$ for the edge device E_e respectively. C represents a set of cloud datacenters C_c. Each C_c is represented as $C_c = \{id_c, h_c\}$ where id_c is the identifier of the datacentre and h_c is the set of host machines $h_{c1}, h_{c2}, ...$ Regardless of the host type i.e. cloud host h_{c_i} or edge host h_{e_i}, each host h_k has hardware h_k^H and software h_k^S capabilities to satisfy the

Table 3.1: Symbol table

Symbol	Description
X	System infrastructure
$\mathcal{D}_i$	An IoT device
δ_i	The data rate of IoT device
E	A set of edge devices
h	A set of host machines
C	A set of cloud datacenters
v	Virtual environment
vm	A virtual machine
cn	A container
P	Processing capabilities
N	The network connection between $\mathcal{D}$, E and C
$\mathcal{S}$	A set of switches
σ	An SDN-controller
A_i	An IoT application
S	Software
H	Hardware
$\mathcal{Q}$	Quality of service
R	Requirements
$\mathcal{P}_i$	Priority
$\mathcal{B}$	The maximum bandwidth available
$T^{d\to\sigma}$	The time taken to transfer the data from IoT device to SDN controller
$T^{\sigma\to e}$	The time taken to transfer the data from SDN controller to edge device
$T^{e\to e}$	The time taken to transfer the data from edge device to edge device
$T^{e\to c}$	The time taken to transfer the data from edge device to cloud
$\mathcal{B}^{ef}$	Effective bandwidth
$count$	The number of IoT devices using the communication channel of the controller
$T^E_{A_i}$	The total transmission time for an application
V	The velocity of propagation of any transmissions
D	The distance between the sender and the receiver
Tp	The propagation time
$T^p_{A_i}$	The total propagation time for an application
T^{P_e}	The processing time of any application microservices
Q	A Queue
$T^Q_{A_i}$	The queuing time of any application
T_{A_i}	The overall execution time for any application
SC_i	The final priority score
$ratio_i$	Compute size from MB to ratio
$size_i$	The IoT application size in MB
λ	The static deciding factor among $\mathcal{P}_i$ and $size_i$
$path$	The channel inside the bandwidth
F_i	A flow
PCT_i	The priority percentage for each path
$pathSize_i$	An amount of data inside the path
$total$	An amount of data inside all paths
B	Bandwidth

requirements of the application. Now, host h_k consists of a set virtual environment $v_{1h_k}, v_{2h_k}, v_{3h_k}, ...$ where, each v_{lh_k} can be either a virtual machine vm or a container cn. Similar to the host h_k, each virtual environment v_{lh_k} also has a hardware specification $v^H_{lh_k}$ and software specification $v^S_{lh_k}$ defined such that $\sum_l v^H_{lh_k} = h^H_k$ and $\sum_k v^S_{lh_k} = h^S_k$. Abstracting the hardware and software processing capabilities as P, we can represent the processing capability of an edge virtual environment as $P^{E_{v_l}}$ and for cloud virtual environment as $P^{C_{v_l}}$. Finally, N represents the network connection between $\mathcal{D}$, E and C and is a subset of $(\mathcal{D} \times E) \cup (\mathcal{D} \times C) \cup (E \times E) \cup (E \times C) \cup (C \times C)$. A set of switches $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ...\}$ and SDN controllers $\sigma = \{\sigma_\mathcal{D}, \sigma_E, \sigma_C\}$ facilitates the network

connectivity in the existing system. An IoT application A_i is defined as a directed acyclic graph (DAG) of microservice $A_i = \{A_i^{\mu_1}, A_i^{\mu_2}, ...\}$ where each $A_i^{\mu_j}$ represents a microservice to execute. Each $A_i^{\mu_j}$ has specific hardware (H), software (S) and quality of service ($\mathcal{Q}$) requirements. Equation 4.1 shows the combined requirements $\mathcal{R}(A_i^{\mu_j})$ for a microservice.

$$\mathcal{R}(A_i^{\mu_j}) = H^{\mu_j} + S^{\mu_j} + \mathcal{Q}^{\mu_j} \tag{3.1}$$

The overall requirement of A_i is given by the sum of requirements of all the microservices as given below.

$$\mathcal{R}(A_i) = \sum_{\forall j} \mathcal{R}(A_i^{\mu_j}) \tag{3.2}$$

At any point of time t, numerous applications $A_1, A_2, ...$ need to be executed on the given infrastructure X. Depending on the type of application A_i, some of them require critical response while others can handle some delay. To allow a smooth execution sequence, a priority $\mathcal{P}_i$ is associated with each application A_i. IoT devices are actively generating data. We consider the IoT device $\mathcal{D}_i$ as a passive entity i.e. it does not process any data but transfers to the edge device. The data transfer happens on a per-second basis, therefore, the total amount of data received by the edge device e_i will also be δ_i multiplied by time t. IoT devices are connected to a switch or an SDN-controller σ which then forwards the data to the respective edge device. Consider the maximum bandwidth available to the IoT device d is $\mathcal{B}_d$ The time taken to transfer the data from the IoT device d to the switch/SDN controller σ can be computed as given in equation 3.3.

$$T^{d \to \sigma} = \frac{\delta_d}{\mathcal{B}_{d \to \sigma}} \tag{3.3}$$

The controller then forwards the data to the respective edge e while consuming $T^{\sigma \to e}$ time. Given the bandwidth of the controller as $\mathcal{B}_\sigma$, it is divided among different communication flows based on how many IoT devices are connected to it. Only an effective bandwidth $\mathcal{B}_{\sigma \to e}^{ef}$ is available for transferring one IoT device's data as given

in equation 3.4.

$$\mathcal{B}^{ef}_{\sigma\to e} = \frac{\mathcal{B}_{\sigma\to e}}{count_t} \tag{3.4}$$

Here $count_t$ is the number of IoT devices using the communication channel of the controller at time t. The time consumed by transferring data from controller to edge device for IoT device d is computed as given in equation 3.5.

$$T^{\sigma\to e} = \frac{\delta_i}{\mathcal{B}^{ef}_{\sigma\to e}} \tag{3.5}$$

Similarly, the data transfer time between edge devices e and between edge and cloud c is computed as given below.

$$T^{e\to e} = \frac{\delta_e}{\mathcal{B}^{ef}_{e\to e}};\quad T^{e\to c} = \frac{\delta_c}{\mathcal{B}^{ef}_{e\to c}} \tag{3.6}$$

Effective bandwidth is computed at each step by the network switch or the SDN controller thus, allowing the data to follow a defined path. For any application A_i, the component microservice A^{μ_i} executes on numerous edge and/or cloud hosts, therefore, the total transmission time for application A_i is given in equation 3.7.

$$T^E_{A_i} = T^{d\to\sigma} + T^{\sigma\to e} + \sum_{\forall e1,e2\in E'} T^{e1\to e2} + \sum_{\forall e\in E',\forall c\in C'} T^{e\to c} \tag{3.7}$$

The propagation time p is computed at the start of all transmissions. Given the velocity of propagation of any transmissions as V, and the distance between the sender and the receiver as D, now, we can calculate the propagation time for the transfer time between IoT device, switch/SDN controller, edge, and cloud as given in the following equations.

$$Tp^{d\to\sigma} = \frac{D_{d\to\sigma}}{V}; Tp^{\sigma\to e} = \frac{D_{\sigma\to e}}{V}; Tp^{e\to e} = \frac{D_{e\to e}}{V}; Tp^{e\to c} = \frac{D_{e\to c}}{V} \tag{3.8}$$

Following the processing happening as given in equation 3.7, the total propagation time for A_i is given in equation 3.9.

$$T^p_{A_i} = Tp^{d\to\sigma} + Tp^{\sigma\to e} + Tp^{e\to e} + Tp^{e\to c} \tag{3.9}$$

Depending on the application A_i, virtual environment $E^{v_{lh_k}}$ of edge device E^k processes the data and sends the processed data to either another virtual environment $E^{v'_{lh_k}}$ on edge or out cloud datacentre. Give the processing capability of an edge and cloud virtual environment, the processing time of any application microservice $A_i^{\mu_j}$ at both edge and cloud host is computed as given below.

$$T^{P_e} = \frac{\mathcal{R}(A_i^{\mu_j})}{\mathcal{P}^{E_{v_l}}}; T^{P_c} = \frac{\mathcal{R}(A_i^{\mu_j})}{\mathcal{P}^{C_{v_l}}} \tag{3.10}$$

Following the processing happening as given in equation 3.7, the total processing time is computed as given in equation 3.11. Here, $E' \subseteq E$ and $C' \subseteq C$ are the edge and cloud hosts executing the application microservice $A_i^{\mu_j}$.

$$T^P_{A_i} = \sum_{\forall e \in E'} T^{P_e} + \sum_{\forall c \in C'} T^{P_c} \tag{3.11}$$

Since, the processing capability of edge/cloud virtual environment v_h is limited, a queue Q_{v_h} is associated with each of them. Data is buffered intermittently while the v_h is busy with the execution. The waiting time for the application A_i in the queue is considered to be the queuing time $T^Q_{A_i}$. The overall execution time for any application A_i is given by the combination of execution, transmission and queuing time as given in equation 3.12.

$$T_{A_i} = T^P_{A_i} + T^E_{A_i} + T^Q_{A_i} + T^p_{A_i} \tag{3.12}$$

3.2.2 Problem definition

Definition: Given a set of IoT applications $A = \{A_1, A2, ...\}$ and the infrastructure $X = \{\mathcal{D}, E, C, N\}$, a suitable deployment solution Δ_m is defined as a mapping for $A_i \in A$ to X ($\Delta_m : A_i \rightarrow X \forall A_i$) if and only if:

1. $\forall A_i^{\mu_j} \in A_i$, $\exists (A_i^{\mu_j} \rightarrow v_h)$ where, $h \in \{h_e \cup h_c\}$
2. $\forall A_i^{\mu_j} \in A_i$, if $A_i^{\mu_j} \rightarrow v_h$, then $H^{\mu_j} \preceq v_h^H \quad S^{\mu_j} \preceq v_h^S$
3. $\sum_{\mu_j} H^{\mu_j} \leq v_h^H$ and $\sum_{\mu_j} S^{\mu_j} \leq v_h^S$

The definition given above considers all the requirements to find a suitable deployment solution. Requirement 1 states that for all the microservices belonging to the IoT application A_i, a mapping must exist between $A_i^{\mu_j}$ and a virtual environment $v_h | h \in \{h_e \cup h_c\}$. Requirement 2 confirms that if a microservice $A_i^{\mu_j}$ is deployed to a virtual environment v_h, the hardware and software requirements of the microservice must be satisfied by v_h. Finally, requirement 3 limits the number of microservices a virtual environment can execute at any time.

The main aim of this research is to find the best solution for all the applications A_i such that the overall execution time T_{A_i} is minimum while the effective bandwidth $\mathcal{B}^{ef}$ is utilized at maximum. In addition to this, the queuing time $T_{A_i}^{Q}$ for the highest priority application $A_{\mathcal{P}}$ should be as low as possible. Given these requirements, we can represent our problem as given below.

$$\textbf{minimize}\, T_{A_i} + \textbf{maximize}\, \mathcal{U}_{\mathcal{B}^{ef}} \tag{4.5}$$

subject to:

$$T_{A_i} \leq T_{A_j} \text{if } \alpha_{A_i} < \alpha_{A_j} \text{and } \mathcal{P}_{A_i} > \mathcal{P}_{A_j} \tag{3.13a}$$

$$\forall i \in A_i, \forall j \in \mu_j \; \exists (A_i^{\mu_j} \rightarrow v_h) \tag{3.13b}$$

Constraint 3.13a specifies that if application A_i arrives before application A_j i.e. $\alpha_{A_i} \leq \alpha_{A_j}$ and the priority of application A_i, $\mathcal{P}_{A_i}$ is higher than the priority of application A_j, $\mathcal{P}_{A_j}$ i.e. $\mathcal{P}_{A_i} > \mathcal{P}_{A_j}$, then the overall execution time for application A_i, T_{A_i} must be less than the execution time for application A_j, T_{A_j}, i.e. $T_{A_i} > T_{A_j}$. Constraint 3.13b states that all the microservices of the application $A_i^{\mu_j}$ should be executed in some virtual environment v_h.

3.2.3 Complexity analysis

The knapsack problem can be used to prove other NP-hard problems by reduction. The knapsack problem is an NP-hard problem that is not solvable in a polynomial time [136]. It is defined as: given a maximum weight capacity W and a set of K items $(0, 1, \ldots, K)$ each having a weight and value of w_i and v_i respectively, maximise the

sum of the values of the items (**maximise** $\sum_{i=0}^{K} v_i\ x_i$) while the overall sum of the weights is less than or equal to the maximum weight capacity ($\sum_{i=0}^{K} w_i\ x_i \leq\ W$) with an item either selected or not ($x_i \in \{0, 1\}$).

Proposition 1: Finding an optimal subset of applications A_i for a given set of application A is an NP-hard problem.

Proof: The knapsack problem as per the previous definition can be transformed, i.e., reduced, into the simplest form of our problem in a polynomial time. The transformation is as follows.

Consider the problem with only single application component $A_i \in A$, change the item's value v_i to $q_i = 1$ and the weight w_i to δ_i and maximum weight W to budget $\mathcal{B}_i$, with parameter x_i remains unchanged. The knapsack problem is already strong NP-hard, thus making our problem $\in$ strong NP-hard.

Inherently, as given in Proposition 1, finding a solution to the knapsack problem in polynomial time leads to finding a solution to our problem in polynomial time. As no such algorithm exists for any NP-hard problem, therefore, we need a heuristic algorithm to find a solution.

3.3 Proposed Framework

To solve the problem specified in section 3.2, we proposed a novel framework that uses two greedy approaches *Multi-queue* and *Bandwidth slicing*. The details are provided below.

3.3.1 Multi-Queues

To reduce the queuing time, we used the concept of *multi-queues* where the waiting queue is divided into a set of priority queues. The principal objective of multi-queues is to dynamically distribute and prioritise the incoming data streams according to a fixed number of queues in the edge and cloud. Specifically, the key procedure involves ensuring that the best queue for each IoT application is selected based on the priority and size of the IoT application. Alg. 1 presents the procedure involved in the solution,

wherein data δ are transmitted from IoT devices and sent to edge devices at a specific time (t).

Algorithm 1: Multi-Queues

```
Data δ_i coming from IoT devices that submitted to edge device E_i within the time interval t.
  Calculate the Score SC_i for each δ_i
SC_i ← using Eq. 3.15
waitingList ← to δ_i //Buffering all δ_i to a waitingList
// Add each δ_i to their specific queue Q_i
for (each Q_i (Q_ls,Q_nm,Q_lt)) do
    for (waitingList) do
        if (δ_i.SC_i = Q_i.value) then
            Q_i ← δ_i
        end
        // now send the δ_i to the execution to be processed starting with Q_ls queue but first
          check if the node has enough CPUs
        if (δ_i.requireCPUs ≤ E_i.currentCPUs) then
            execution ← δ_i waitingList ← remove δ_i
        end
    end
end
```

Subsequently, the first step is the computation of the Score SC for each IoT application A_i, where the Score SC is the final priority score that will be used to divide the data δ in the queues. Thus we need to find the $ratio_i$ for each δ_i using equation 3.14.

$$ratio^{P_A} = \frac{size^{P_A}}{\sum_i^j size^{P_A}} \tag{3.14}$$

Where $ratio_i$ is the process of converting the $size_i$ that has been provided by the user from MB to $ratio_i$, where $ratio_i \, \epsilon \, \{0, 1\}$, and $size$ is the IoT application A size in MB. Next, to separate the data to the queues we need to find the Score SC_i for each IoT application A using equation 3.15:

$$SC_i = \mathcal{P}_i \times \lambda + (ratio_i \times (1 - \lambda)) \tag{3.15}$$

Where, λ is a static deciding factor among the priority $\mathcal{P}_i$ and δ_i size of the IoT application A_i, where SC_i and priority $\mathcal{P}_i \, \epsilon \, \{0, 1\}$, and $\lambda = \{0.8\}$. The Score SC results comes in three types, low priority where the $SC \, \epsilon \, \{0.1, 0.3\}$, normal priority where the $SC \, \epsilon \, \{0.4, 0.6\}$, and high priority where the $SC \, \epsilon \, \{0.7, 0.9\}$. For example, if we have $\mathcal{P}_i = 0.9$, and $ratio_i = 0.5$, then the Score $SC_i = 0.8$, which means that it is a high

priority and should be forwarded to the latency-sensitive queue that we will discuss next. Then, we buffered all the data δ and their Scores SC in the $waitingList$ (Lines 2 - 6). In the second step, each δ is added to the appropriate queue Q depending on their SC. Specially, we have three types of queues Q, latency-sensitive Q_{ls}, normal Q_{nm}, and latency-tolerant Q_{lt}. Last step, we send the δ to the $execution$ to be processed, starting with Q_{ls}, Q_{nm}, then Q_{lt}, according to the $currentCPUs$ in the edge device.

3.3.2 Bandwidth Slicing

Bandwidth slicing is primarily designed to slice the bandwidth statically between the *paths*, where *paths* is the channels inside the bandwidth. The procedure aims to determine the best slicing percentage for the bandwidth based on the priority and data size of each application.

Algorithms 2 and 3, illustrated in this chapter, are key components of the bandwidth-slicing framework. Algorithm 2 is used in the initial stage to classify and prioritise data streams as they are received. It is responsible for receiving network flows, computing a score for each flow, and sorting these flows into queues based on their scores. This process ensures that flows are appropriately categorised according to their respective priorities or characteristics. Algorithm 3, on the other hand, is applied later to manage and allocate bandwidth dynamically based on these priorities. It operates at a subsequent stage where it manages the allocation of bandwidth among these queues. Depending on the type of priority assigned to each queue, Algorithm 3 slices the available bandwidth accordingly, ensuring that higher-priority queues receive the bandwidth they require.

Moreover, Algorithm 2 is applied within the framework as part of the global network agents deployed in SDN and SD-WAN controllers. These controllers are responsible for data stream scheduling, where Algorithm 2 is used to classify and prioritise incoming data flows based on their characteristics (such as size and priority). This classification is crucial for determining how resources are allocated in the network. Algorithm 3 is then applied within the same framework, specifically focusing on bandwidth slicing. Once the data flows have been prioritised by Algorithm 2, Algorithm 3 is responsible for dynamically slicing the available bandwidth among the different priority queues.

This ensures that each IoT stream is allocated the appropriate amount of bandwidth according to its priority, thereby optimizing the use of network resources. Together, these algorithms complement each other within the bandwidth-slicing framework.

In detail, the first stage is the receipt of flows F that is sent to either edge devices or the cloud, where, each F contains a packet that includes one δ_i from one IoT application A_i. After that, the SC for each F is computed using equation 3.15 and buffered in the $flowList$ (Lines 7-11). In order to slice the bandwidth, the number of $paths$ must be known. This is determined by checking the priorities of all the F stored in the $flowList$ and identifying the number of $paths$ (Lines 13-22).

Algorithm 2: Bandwidth slicing

Input: ls, nm, lt: priority types, $total$: number of flows, $availableBw$: available bandwidth, $usedBw$: used bandwidth, $weightedAverage$: compute the average between multiple $paths$

```
1  Received flows F contains δ to be sent to node E_i.
2  Calculate the Score SC for each flow F
3  SC_F ← using Eq. 3.15
4  flowList ← to F //Buffering all F to a flowList
5  // Count the types of paths
6  for (flowList) do
7      path_i ← F_i.SC_i
8      switch path do
9          case 0 do
10             lt ← path
11         end
12         case 1 do
13             nm ← path
14         end
15         case 2 do
16             ls ← path
17         end
18     end
19 end
20 total=flowList.size
21 slicing()
```

In the next stage $slicing()$ procedure is applied as per the details in Alg. 3, whereby the slicing of the bandwidth is based on the number of available $paths$. There are two types of slicing, the first takes place when there is only one type of $path$ (e.g. lt, nm, or ls), after which the entire bandwidth is given to that $path$ (Lines 4-10). The second type of slicing occurs where there is more than one type of $path$ (e.g. lt and nm, or lt and ls, or ls and nm, or lt, nm, and ls). Subsequently, the $weightedAverage$ for each $path$ is calculated using equation 3.16 and multiplied by the $availableBw$. After

Algorithm 3: Slicing

```
1  if (total==0) then
2  |  usedBw = 0
3  end
4  else
5  |  switch (lt, nm, ls) do
6  |  |  case (lt ! = 0) do
7  |  |  |  usedBw = availableBw / lt
8  |  |  end
9  |  |  case (nm ! = 0) do
10 |  |  |  usedBw = availableBw / nm
11 |  |  end
12 |  |  case (ls ! = 0) do
13 |  |  |  usedBw = availableBw / ls
14 |  |  end
15 |  |  case (lt & nm ! = 0) do
16 |  |  |  weightedAverage_{lt,nm} ← using Eq. 3.16
17 |  |  |  usedBw = availableBw * weightedAverage_{lt,nm}
18 |  |  end
19 |  |  case (lt & ls ! = 0) do
20 |  |  |  usedBw = availableBw * weightedAverage_{lt,ls}
21 |  |  end
22 |  |  case (ls & nm ! = 0) do
23 |  |  |  usedBw = availableBw * weightedAverage_{ls,nm}
24 |  |  end
25 |  |  case (lt & nm & ls ! = 0) do
26 |  |  |  usedBw = availableBw * weightedAverage_{lt,nm,ls}
27 |  |  end
28 |  end
29 end
```

this, the bandwidth is divided among the *paths* in line with the *weightedAverage* for each *path*, with the largest percentage of the bandwidth being allocated to the *ls path*, followed by the *nm path* and the *lt path* (Lines 11-24).

$$weightedAverage = \sum_{i}^{j} \frac{\mathcal{PCT}_i * pathSize_i}{total} \tag{3.16}$$

equation 3.16, shows the weighted average for each *path*. Where i and $j \; \epsilon \{ls, nm, lt\}$, and $\mathcal{PCT}_i$ is the priority percentage for each *path* that will be defined by the user. Then we have a *path* size that clarifies how many δ_i inside it is represented by $pathSize_i$. Lastly, we have *total* that represents the total number of δ_i inside all *paths*.

$$\mathcal{NU}\% = \frac{size_i * 100}{availableBw * \Delta t} \tag{3.17}$$

equation 3.17, shows the network utilisation for each *path*, where $size_i$ in bits is mul-

tiplied by 100, and divided by $availableBw$ multiplied by the Δt time interval.

3.4 Evaluation

In this section, we evaluate our proposed work on a self-driving car test case.

3.4.1 Experiment Set-up

3.4.1.1 Test Case

Figure 3.2 illustrates the fundamental framework used in integrating autonomous vehicles. Modern self-driving cars are equipped with a plethora of devices such as sensors, cameras, radars, and mechanisms for managing speed. These devices share their collected data with the SDN controller, which is positioned within the proximity of low-latency 5G networks. The controller then assesses this information to manage the data's routing and prioritisation. Furthermore, SD-WAN plays a critical role in ensuring the efficient flow of network traffic to and from autonomous vehicles, thereby facilitating the evolution of these vehicles into entities that are both more intelligent and safer [137]. Data centres, located in various parts of the city, both at the edge and in the cloud, receive data from the vehicles for processing on their host machines. For further analysis, this data is transmitted to other host machines across different data centres via the controllers, which in turn, provide feedback for real-time decision-making. Additionally, the network supports communication between different edge and cloud data centres through the SD-WAN, while also ensuring that the application's response times and processing needs are adequately met.

In this scenario, the IoT devices in a car capture raw data and assign it a priority level. Each piece of data is then ranked according to its priority and transmitted to an edge data center. Upon arrival at the edge data centre, the data packets are organised and placed into separate queues based on their priority and size before being forwarded to edge devices for processing. Subsequently, the data is transmitted to cloud data centres via SD-WAN on 5G towers for additional processing. The SD-WAN controller manages the data by selecting the optimal and quickest path and adjusting the bandwidth to accommodate the data size. Similar to the process at the edge data centre, the cloud

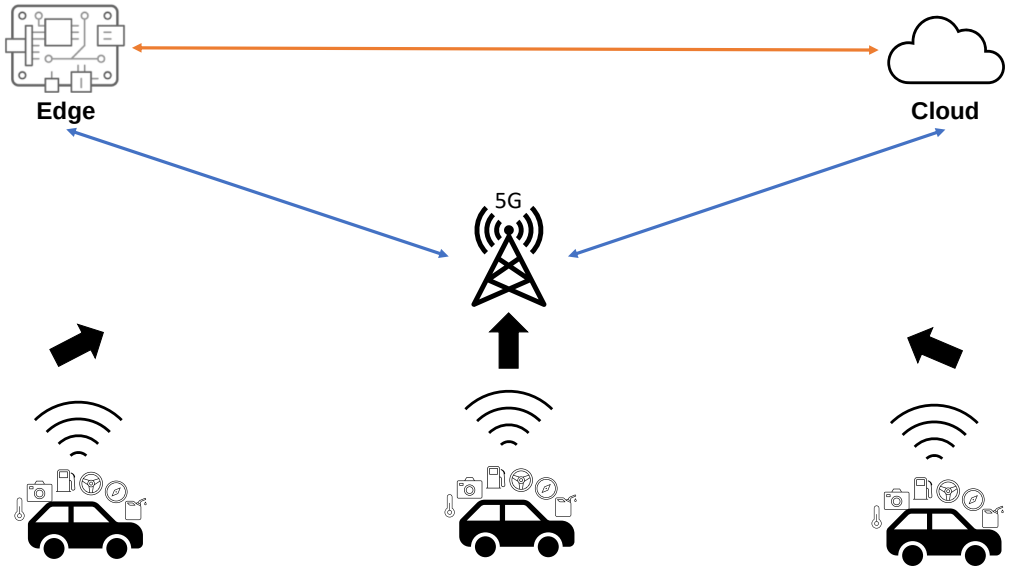


Figure 3.2: Data transfer and processing in Self-driving cars

data centre organises and queues the data according to its priority and size, preparing it for processing by the VMs. Figure 3.3 shows a detailed illustration of the whole process.

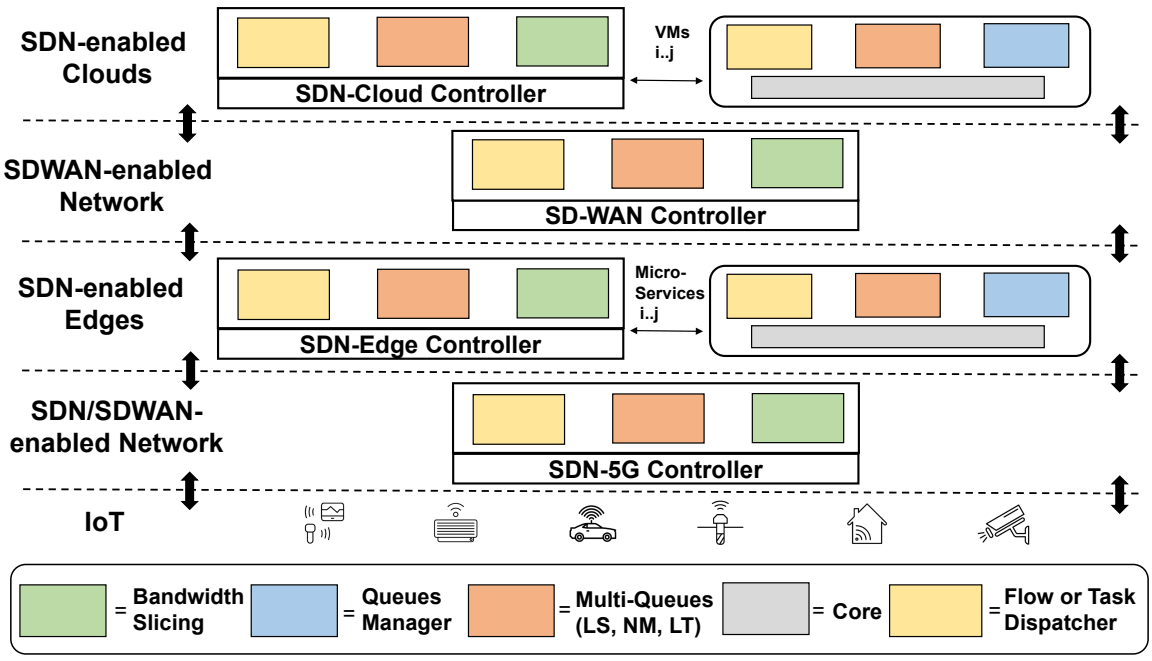


Figure 3.3: Scenario process in our IoT-edge-cloud environment

3.4.1.2 Configuration

We model the scenario using the open source simulator *IoTSim-Osmosis* [138]. Table 3.2 shows the specific configuration details for the given test case. We vary the number of IoT devices from 10 to 60 for the given test case. The details about the number of devices are given in Table 3.3. We compared our results with two approaches: the First Come, First Serve (FCFS) and Shortest Job First (SJF) methods.

Table 3.2: Test case configuration

IoT Device		**Edge Device**		**Host(Edge)**		**VM(Cloud)**	
IoT type	**car**	**Edge type**	**Raspberry Pi**	**Storage**	**640 GB**	**Storage**	**10 GB**
Max BW	**100 Mbps**	**Max BW**	**100 Mbps**	**Max BW**	**10000 Mbps**	**Max BW**	**1000 Mbps**
Required CPUs	**10**	**Pes**	**10**	**Pes**	**4**	**Pes**	**4**
Network	**5G**	**RAM size**	**10000**	**RAM size**	**32000**	**RAM size**	**512**
Max battery cap	**100 mAh**	**MIPS**	**250**	**MIPS**	**1250**	**MIPS**	**250**

3.4.2 Experiment results

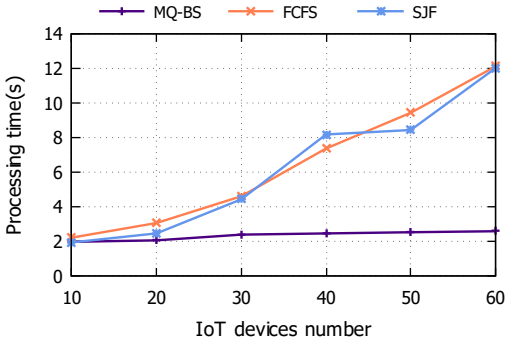
This section presents the results of our proposed MQ-BS approach. Figure 3.4a shows the average processing time of each test as compared to the FCFS and SJF. As shown in the Figure, our proposed approach achieves an average gain of 71% as compared to FCFS and 73% compared to SJF. The trend is also followed for the transmission time with 49% savings as compared to the FCFS and 74% with SJF as shown in Figure 3.4b. The trend is also followed for the queue waiting time with 164% savings as compared to the FCFS and 98% with SJF as shown in Figure 3.4c. Table 3.4 shows a comparison of the results in detail between FCFS, SJF, and our proposed MQ-BS policies.

3.4.2.1 Scalability result

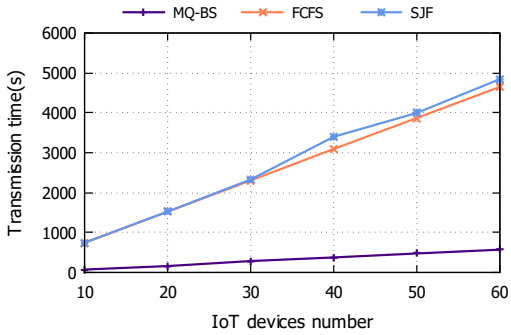
Figure 3.5a shows the average simulation time of each test as compared to the FCFS and SJF. As presented in the Figure, our approach achieves an average gain of 143%

Table 3.3: Infrastructure device configuration

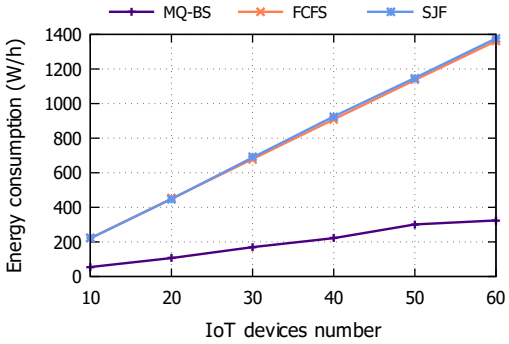
Number of IoT Devices	**Number of Edge Devices**	**Number of hosts**	**Number of VMs**
10-60	**2**	**2**	**2**



(a) Processing time



(b) Transmission time



(c) Queue waiting time

Figure 3.4: The experiment results

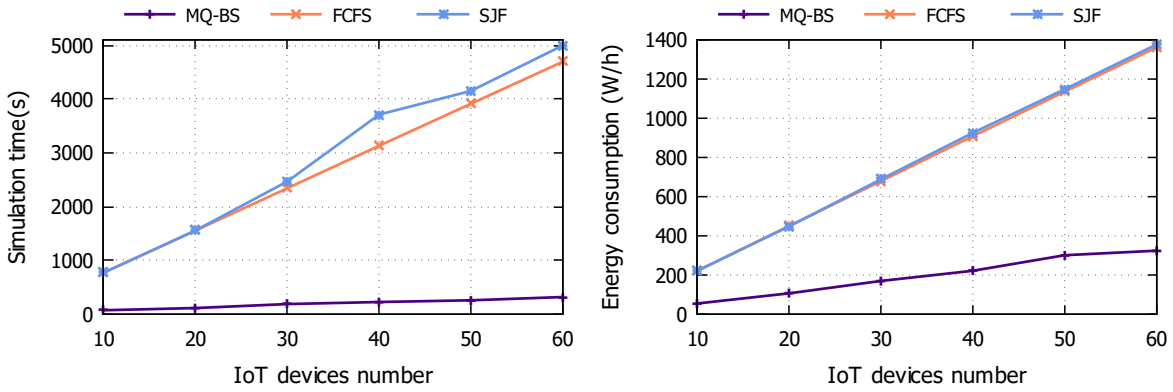
Table 3.4: A comparative table for the results

Processing Time				Transmission Time			Queues Waiting Time		
IoT Device	**MQ-BS**	**FCFS**	**SJF**	**MQ-BS**	**FCFS**	**SJF**	**MQ-BS**	**FCFS**	**SJF**
10	**1.9**	**2.21**	**1.93**	**73**	**748**	**749**	**36**	**105**	**710**
20	**2.05**	**3.06**	**2.46**	**160**	**1526**	**1527**	**88**	**474**	**1427**
30	**2.38**	**4.60**	**4.42**	**286**	**2308**	**2331**	**140**	**978**	**2140**
40	**2.45**	**7.36**	**8.17**	**375**	**3095**	**3401**	**192**	**1713**	**2858**
50	**2.52**	**9.43**	**8.43**	**475**	**3870**	**3996**	**240**	**2368**	**3571**
60	**2.58**	**12.11**	**12.01**	**571**	**4648**	**4842**	**292**	**3259**	**4284**

as compared to the FCFS and 149% compared to SJF. Finally, Figure 3.5b shows the average energy consumption of each test as compared to the FCFS and SJF. As presented in the Figure, our approach achieves an average gain of 24% as compared to the FCFS and similar 24% compared to SJF.

In summary, the system being proposed demonstrates substantial advancements over FCFS and SJF methodologies in both edge and cloud computing environments. It enhances processing efficiency by up to fourfold and reduces the data transmission duration from IoT devices to the cloud through edge computing and SD-WAN by as much as ninefold. Additionally, beyond the significant enhancements in data process-

ing and transmission speeds, it's observed that the system's innovative policies also contribute to a threefold reduction in energy consumption. Notably, the benefits in terms of both time and energy savings increase proportionally with the amount of data processed.



(a) Simulation time (b) Energy consumption

Figure 3.5: Scalability results

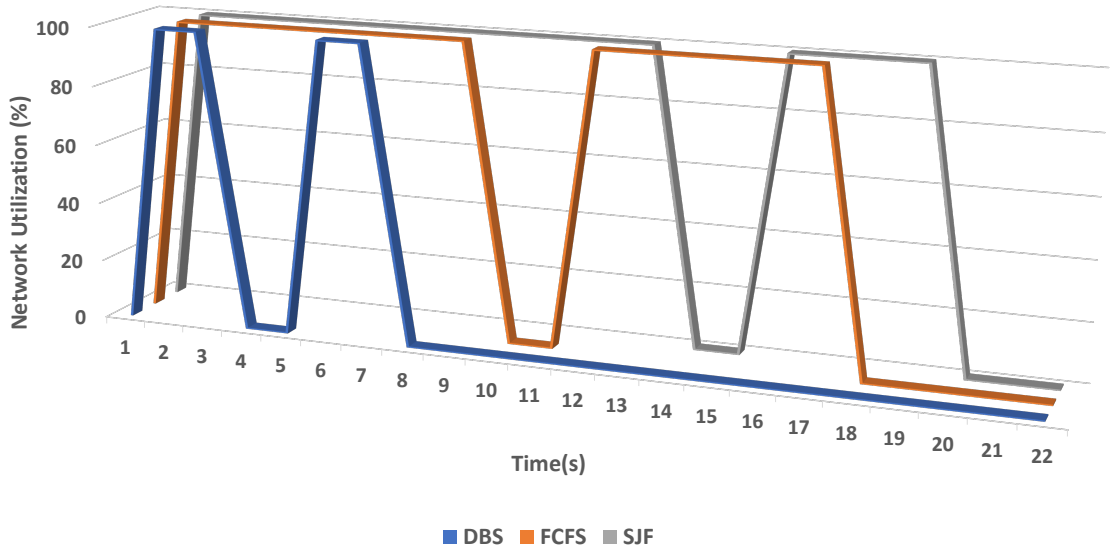


Figure 3.6: Comparing the network utilisation for the three policies

3.4.3 Network Utilisation

This section describes the network utilisation measurement results for all systems from the start to the end of the simulation using equation 3.17. Figure 3.6 shows the network

utilisation percentage for FCFS, SJF, and MQ-BS policies. It can be seen that at the beginning, it started to use 100% of the network when it was sending data from IoT devices to microservices. Then immediately after that, it drops to 0% because the data had arrived at destination microservices and started the processing phase. Next, 100% was used from the network, because microservices started to send the data to the cloud. Finally, it drops again to 0% because the data had arrived at the destination VMs and started the processing phase. However, our proposed system shows the same way of using the network as in the previous systems but it decreases the overall time of network usage. So, this illustrates that our system improved the time of network utilisation by up to 7 times and 7.5 times compared with the FCFS and SJF systems, respectively.

3.4.4 Auto-Adaptation

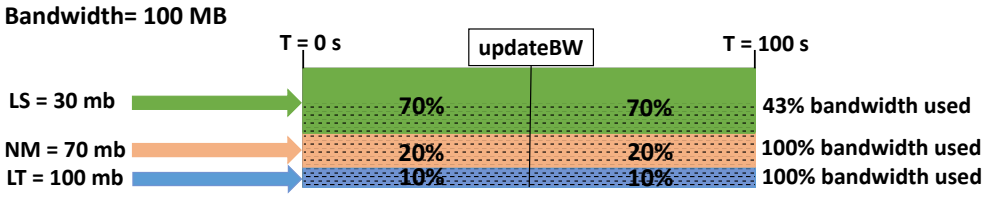
Although the results so far show promising optimal performance, sometimes bandwidth static slicing can lead to a degradation in the network utilisation. Figure 3.7a shows an example of how such problem might arise. Note that set-up and configuration is similar to the previous experiment but with only 10 IoT devices. The Figure 3.7a has 100 MB of bandwidth where it is sliced/divided into three parts: 70% is assigned to the latency-sensitive (ls) *path*, 20% is given to the normal (nm) *path*, and 10% is assigned to the tolerant-sensitive (lt) *path*. Suppose that the ls *path* receives 30 MB of data every second, nm *path* receives 70 MB of data every second, and lt *path* receives 100 MB of data every second (as shown in the Figure). If the ls *path* is only using 30 MB per second, then 40% of its sliced network would be wasted. As such, this chapter contributes to solving this problem by proposing an auto-adaptive network slicing algorithm. The algorithm is designed to dynamically tune the network slicing percentage based on the network utilisation of each *path*, as shown in Figure 3.7b.

$$pathRatio = \frac{pathFlows_i}{total} \tag{3.18}$$

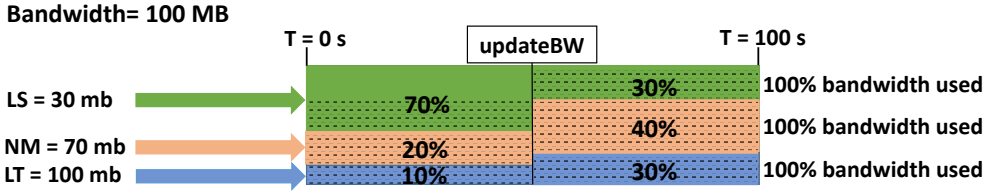
equation 3.18, shows the $pathRatio$ of each path. Where $pathFlows_i$ is the F numbers of q_i, and q_i is one of our proposed paths (ls, nm, lt), divided by the $total_i$ number of

flows in that *path*.

Every *path* receives multi flows every second, depending on the data coming from IoT Devices. So, after computing the number of flows that are used in every *path*, we use it in our Alg. 4 to calculate the new percentage for every *path* every time the bandwidth is updated in the system.



(a) Without Auto-Adaptation



(b) With Auto-Adaptation

Figure 3.7: The Auto-Adaptation example

Algorithm 4: Auto-Adaptation

Input: $minPCT$: The minimal percentage for Auto-Adaptation, $newWeightedAverage_i$: The $weightedAverage_i$ with the new $\mathcal{PCT}_i$, $oldWeightedAverage_i$: The $weightedAverage_i$ that computed in Alg.3

1 $pathRatio_{lt,nm,ls} \leftarrow$ using Eq. 3.18 // Measures the network utilization $\mathcal{NU}$ for all *paths*
2 **if** $(pathRatio_i \geq minPCT)$ **then**
3 $\quad usedBw = availableBw * newWeightedAverage_i$
4 **end**
5 **else**
6 $\quad usedBw = availableBw * oldWeightedAverage_i$
7 **end**
8 **return** $usedBw$

The main goal of auto-adaptation is to dynamically allocate the percentage of *paths* in the bandwidth slicing mechanism. Thus, the procedure seeks the optimal slicing percentage for the bandwidth based on the network utilisation $\mathcal{NU}$ for each *path*. Alg. 4 clarifies the auto-adaptation procedure, which starts by measuring the $\mathcal{NU}$ for each *path* as per equation 3.17. Following this, the resulting percentage *pathRatio* is compared with the minPCT defined by the user. If the *pathRatio* is equal to or bigger than the $minPCT$, the new *pathRatio* is employed in the *weightedAverage*

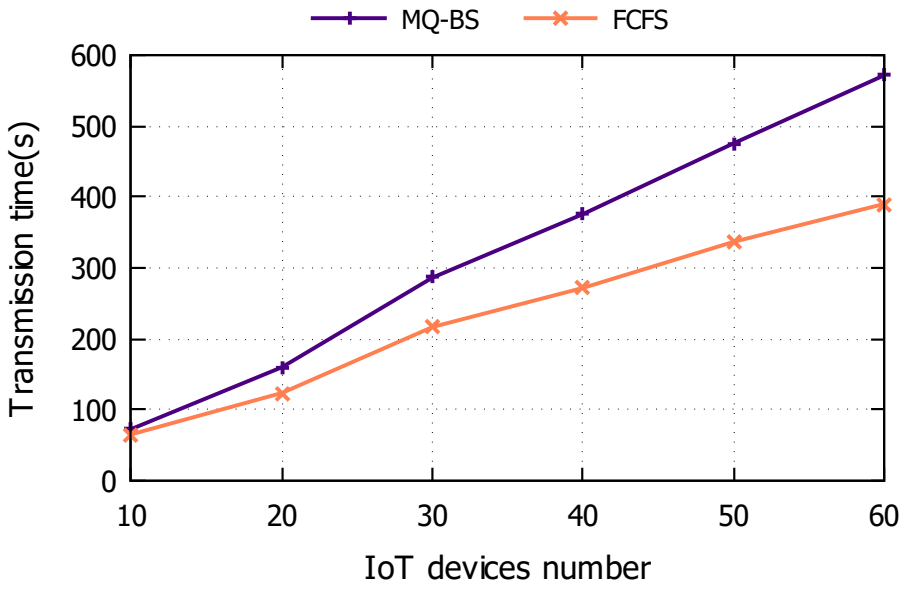


Figure 3.8: Auto-Adaptation transmission time

using equation 3.16 to comprise an improved percentage that improves the bandwidth slicing between the *paths*. If the *pathRatio* is smaller, the static percentage that was previously employed to the *path* will be utilised. Figure 3.8 shows the results of a comparison between the MQ-BS system and the MQ-BS system with the Auto-Adaptive network slicing algorithm, which showed an improvement in the transmission time of 46%.

3.5 Further Evaluation and Validation

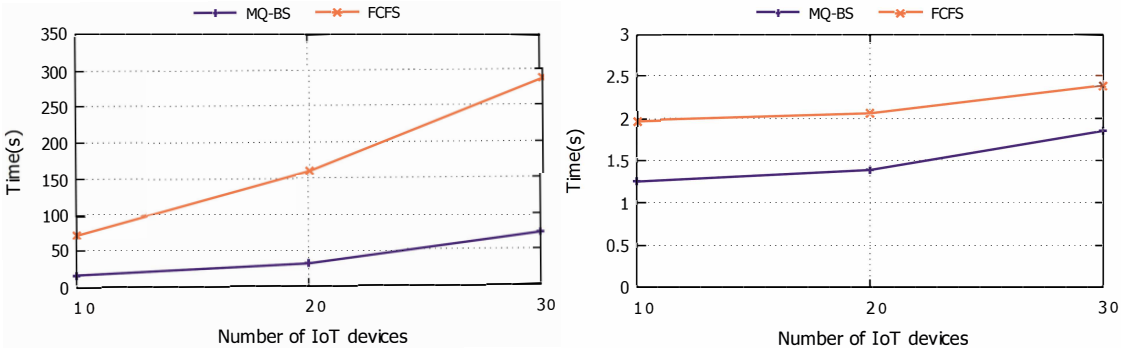
Our research not only puts our proposed system to the test in a simulated setting but also verifies its performance in a practical IoT-oriented SDN environment. For real-world testing, we utilised actual edge computing devices, including three Raspberry Pi units, an SDN-capable switch, and a laptop. To simulate IoT devices and create IoT data, we employed sensor simulators. The setup involved running a sensor simulator on one Raspberry Pi, an edge processing simulator on the second, and a virtual machine (VM) on the third Raspberry Pi. Each Raspberry Pi was equipped with a 1.4GHz quad-core processor and 1GB of RAM. In terms of networking, we implemented an Open vSwitch (OvS) on a Linux-based switch powered by an Intel N3700 Processor with 8GB RAM. Additionally, a Ryu controller, serving as the SDN

controller, was operated on the laptop, which featured an Intel i7-8565U quad-core processor at 1.99GHz and 16GB of RAM.

Workload and Dataset: Our study employs a dataset from a real-world smart building Urban Observatory, Newcastle University[139] to create a realistic workload scenario. This dataset includes data samples collected from various sources such as temperature, NO2, and gas sensors. To facilitate data exchange between devices, we utilised the MQTT protocol. We also introduced a multi-queue policy on both the edge emulator and the VM to prioritise data based on the sensor's importance. Additionally, we implemented a bandwidth slicing strategy within the SDN controller to efficiently manage network bandwidth among the devices.

Methodology: We have developed three test applications, incorporating sets of 10, 20, and 30 sensors. Initiating with the sensor emulator, we configure it to generate an input flow of 10 to 30 records per second, which are subsequently dispatched to the edge layer. Upon arrival, the data goes through the organisation via a multi-queue policy for processing at the edge. Following this phase, the processed data is transferred to the VM through a network switch. This switch routes the data to the SDN controller, which then allocates routing paths and bandwidth slices based on data priority before forwarding the data to the VM for additional analysis. The VM undertakes the final sorting and processing of the data.

Results: We calculated the average time it takes for data to be transmitted from the sensor, through the switch, and finally to the VM for all three applications. This was then compared with the average transmission times obtained from simulated experiments. Furthermore, we assessed the average time required to process data at the edge and within the VM for each application, comparing these figures against those derived from simulation experiments. As depicted in Figure 3.9, there is a noticeable increase in both processing and transmission times with the addition of more sensors. This trend suggests a positive relationship, underscoring that the precision and reliability of our simulation findings are in alignment with those observed in an actual IoT-based SDN setup.



(a) Average transmission time (b) Average processing time

Figure 3.9: Validation results

3.6 Related work

There has been significant previous research in areas related to cloud task offloading, IoT stream processing, bandwidth slicing, and congestion control. In mobile computing, offloading tasks generally means transferring them to the cloud, and it's important to understand how different computing resources affect data transmission. The LEO[88] system has made advancements in reducing energy usage by managing various sensor processing tasks on mobile devices, yet it does not address the variability in IoT networks. Similarly, MAUI[140] overlooks the latency introduced during the transfer of data from the edge to the cloud, despite recognising the variability in resources. Therefore, our system presents an opportunity to enhance these networks by addressing these gaps.

Improvements in edge computing have shifted the processing of cloud-based data closer to its source, significantly decreasing the delay in processing. The author in [90] introduced a system designed for edge-based stream processing capable of handling data from various IoT devices simultaneously. However, the primary focus of this system is to improve adaptability to fluctuations in wireless network environments, rather than solving problems related to bandwidth division. Additionally, [91] focuses on improving the efficiency of analytical tasks, paying less attention to the queuing delays that occur during stream data processing.

NebulaStream[84] is a system that channels data streams to various processing tasks for specific data-flow programs through APIs. However, it lacks the capability to dis-

tinguish between the latency sensitivities of different IoT applications, thus failing to handle queue delays effectively. In contrast, our suggested system offers an efficient traffic scheduling solution, ideal for use across IoT-edge-cloud continuum environments, particularly suitable for scenarios with diverse data record types and distinct QoS demands. In networking, SDQ[92] introduces a method for selecting optimal queues and routes for each incoming flow, aiming to reduce network workload disparities. Nevertheless, it does not account for the cloud network and bandwidth slicing considerations. On a related note, NS[83] presents a communication solution based on network slicing, yet it overlooks the aspects of bandwidth slicing, as well as edge and cloud processing.

Congestion management is a widespread approach within the network community, typically implemented by controlling the rate of transmission and directing network packets to their destinations. The QJUMP[86] framework, facilitates the routing of messages to different queues based on their priority levels, a concept akin to our system's multi-level queue management functionality. Nonetheless, QJUMP lacks support for applications that process streaming data. Other systems focused on receiver-driven flow control, such as Homa[89], pHost[87], and NDP[85], have been shown to effectively diminish the latency for small messages. These systems, however, rely on switch-based solutions that presuppose equal ingress and egress throughputs a premise not valid within the IoT-edge-cloud spectrum. Our system innovatively merges dynamic bandwidth management with comprehensive traffic coordination at the application level, offering the adaptability required for controlling throughput and adjusting bandwidth in real time during streaming activities. The detailed properties of recent and our proposed systems are compared in Table 5.1.

3.7 Conclusions and future work

This article introduces an innovative traffic scheduling system that operates on a distributed basis and prioritises QoS through multi-level queue management. The aim of this system is to enhance overall throughput while reducing queue delays and improving QoS for applications sensitive to latency. Our approach categorises incoming traffic based on their latency requirements into multiple levels of queues and employs

Table 3.5: Comparison of various scheduling systems with the one proposed

Systems	**Features**								
	Cloud processing	**SDN support**	**Auto Adaptation**	**BW slicing**	**Stream processing**	**Queuing delay**	**Edge processing**	**IoT devices**	**Latency**
LEO [88]	✓					✓			✓
MAUI [140]	✓				✓	✓	✓		✓
Frontier [90]			✓		✓	✓	✓	✓	✓
Approxiot [91]			✓		✓		✓	✓	✓
Nebulastream [84]	✓				✓			✓	✓
Homa [89]	✓					✓			✓
pHost [87]						✓			✓
NDP [85]						✓			✓
SDQ [92]		✓				✓	✓		✓
NS [83]		✓		✓		✓			✓
QJUMP [86]	✓					✓			✓
Proposed	✓	✓	✓	✓	✓	✓	✓	✓	✓

bandwidth slicing to allocate network bandwidth accordingly. This bandwidth allocation is dynamically adjusted in sync with real-time network usage analysis. Utilising these techniques, our system significantly improves latency and throughput in edge-cloud computing scenarios. Comparative results demonstrate that our system reduces processing latency by up to four times and network latency by up to nine times, outperforming traditional methods such as FCFS and SJF. Furthermore, it achieves a threefold reduction in energy consumption for both edge and cloud computing environments and the network. Future research will explore more sophisticated algorithms to optimise bandwidth slicing for enhanced performance.

4

Dynamic Data Streams for Time-Critical IoT Systems in Energy-Aware IoT Devices Using Reinforcement Learning

Contents

Summary

This chapter discusses the deployment of thousands of sensors conscious of energy usage across various environments like manufacturing, control mechanisms, disaster relief, and flood prevention. These scenarios demand solutions that are not only timely but also energy-efficient to prolong the lifespan of the sensors. It introduces a strategy based on RL for dynamic data handling in time-sensitive IoT systems, specifically focusing on energy-efficient IoT devices. Utilising the Q-Learning technique, this approach is capable of modulating the data transmission rate in accordance with the availability of renewable energy sources. This ensures both the reliability of data collection and the conservation of sensor battery life. An evaluation of this method using past solar radiation data revealed that it could enhance data transmission by as much as 23% over other profiles considered, guaranteeing the device's uninterrupted function.

4.1 Introduction

The IoT is a concept beginning to be a natural element of human development and technological progress. IoT devices are used in many areas of everyday life, including smart homes, factories and cities [123]. IoT devices are also used in time-critical systems, i.e., where it is essential to obtain data processing results in the shortest possible time [11]. Examples of such systems are various solutions used during natural disasters, such as fires or floods. The key factor of such systems is the processing of up-to-date, non-delayed data from sensors installed in IoT devices. To achieve this, devices should be ready to transmit a data stream with necessary requirements.

Unfortunately, transferring a significant amount of data from sensors is associated with a high demand for energy to make the measurements and then send the data to the edge and computing clouds. However, this can be difficult to achieve for IoT devices with limited computing and power resources. Especially when they are powered by renewable energy sources such as solar energy. However, the device can respond to changes in the availability of renewable energy by changing the frequency of collecting

and transmitting measurement data. The chapter proposes dynamic data streams, which can be changed to consume the device's available resources accordingly.

Nevertheless, dynamic data streams in time-critical systems are related to the intelligent management of their parameters. Therefore, we propose to use the concept based on autonomic computing [141] to manage their parameters through dedicated management agents that can monitor and plan adaptation actions. It is also important that the purpose of device adaptation may depend on the system's operating goal. For example, in a flood risk situation, the system should work with the most up-to-date data possible, paying less attention to maintaining the system's lifetime. On the other hand, during normal operation, the system should strive to maintain as long a lifetime as possible to prepare for emergency situations. The implementation of the autonomic computing concept in IoT devices is complex since they have very limited resources. Therefore, we propose the usage of cooperating osmotic agents associated with devices and the edge datacenters [142]. The agents operating on the devices send data regarding the device operation, e.g., battery level, current configuration, while the edge agent plans device reconfiguration actions, which are then sent in response to be executed on the devices.

The agent's logic could be implemented in the form of decision rules specifying actions that will be performed in specific situations. However, it would require having a particular model of the device and the environment in which it works. Therefore, in the chapter we propose to implement the agent's logic based on reinforcement learning. It is used in systems where, based on the observation of the system operation and the actions taken, their effectiveness in the form of a reward can be assessed.

Overall, this chapter tries to solve the research question:

How to do real-time data generation on the IoT while optimising the energy of resource-constrained devices in time-critical systems on the IEC continuum?

To address the research question, this chapter introduces a solution based on RL, called dynamic data streams, designed for energy-conscious IoT devices in time-sensitive IoT systems. This proposed mechanism can adapt the data transmission rate according to the availability of renewable energy resources, ensuring consistent data collection

while also considering the lifespan of the sensor battery.

In summary, the following are the chapter's primary contributions:

- we formulated the limitation of the power resources problem in the IoT device,
- we proposed reinforcement learning-based dynamic data streams for time-critical IoT systems in energy-aware IoT devices,
- we evaluated our proposed approach performance using a levee monitoring system in river flood scenario.

The chapter is organised as follows. The second section deals with an overview of state of the art. The third section discusses the motivation, while fourth describes the formal model and problem definition. Section five presents the proposed concept based on reinforcement learning, which is then evaluated in section six. Finally, the chapter is summarized and future work is discussed.

4.2 Related Work

The extensive integration of RL with IoT devices is well-documented in scholarly articles. A prime example is found in the work [143], which showcases various instances where RL has been applied within IoT settings. IoT devices are capable of implementing adaptation strategies across multiple system layers. For instance, within the perception layer of a smart vehicle, decisions regarding speed, direction, and obstacle avoidance can be made. At the application layer, particularly in edge or cloud servers, it involves making choices on task scheduling, data caching, or allocation of virtual machine resources. Moreover, RL algorithms play a crucial role in managing the bandwidth or the data transmission rate in the network layer.

RL techniques can enhance the efficiency of tasks related to sensor coverage, as noted in [144]. These methods allow a network of sensors to cover as vast an area as possible while minimising energy usage. The premise is that extensive coverage by individual sensors often leads to significant energy drain. However, in systems comprising multiple agents where devices interact with one another, RL can refine the operations of

these devices specifically for sensing tasks, thereby reducing total energy consumption. One strategy involves adapting the Q-Learning algorithm, for instance, through the implementation of a distributed value function as proposed by [145]. This adaptation accelerates the learning process within distributed settings.

In the study by [146], three strategies for maximising data transfer from an IoT device with limited battery life are examined. Two strategies involve online/offline optimisation, operating under the assumption of foreknowledge regarding future energy availability and environmental conditions. Conversely, the RL strategy, more applicable in real-world scenarios, has only sporadic knowledge of energy levels and system states. Over time, the RL strategy demonstrates performance on par with the optimisation strategies. In a separate study by [147], RL is applied to manage data transmission from battery-powered devices to a single base station over constrained channels. The objective was to optimise data bandwidth to the base station without depleting the device batteries. The base station selects devices, gathers their state information, and then uses this data to determine an action based on a pre-learned policy, which is then communicated back to the devices. Given the extensive action and state spaces involved, Deep RL algorithms incorporating Long Short-Term Memory (LSTM) were utilised for calculating Q-values.

In the context of Mobile Edge Computing (MEC), RL is employed to optimise processor frequency settings, as discussed by [148]. This approach involves the deployment of an RL algorithm on an edge server, aiming to enhance the efficiency of processing incoming requests. Upon receiving a request, the server evaluates the current CPU load and battery condition to determine the feasibility of processing the request. If the decision is affirmative, the CPU's operational frequency is increased, which, although consumes more energy, leads to improved processing times. Experimental outcomes demonstrate that the server, through the application of RL, becomes adept at managing various request sizes under different conditions, such as battery levels, outperforming traditional rule-based methods like best/worst fit strategies or other algorithmic solutions including sliding window techniques.

An integration between RL and LSTM neural networks is proposed for situations where an RL agent needs to make a decision on activating sensors for real-time data collection

or relying on past data for predictions, as described in [149]. This method aims to maintain an equilibrium between minimising energy use and ensuring the precision of data collected. Both the RL agent and the LSTM network undergo pretraining, and their performance is assessed through a comparison of model-free and model-based RL strategies. The findings indicate that these algorithms are effective in deactivating sensors without compromising data accuracy, thereby extending battery life.

4.3 Motivation

Consider the levee monitoring system installed beside the river, as depicted in Figure 5.1. This system is designed to identify potential conditions that could lead to the levee's failure during flooding events, thereby preventing extensive property damage and loss of life.

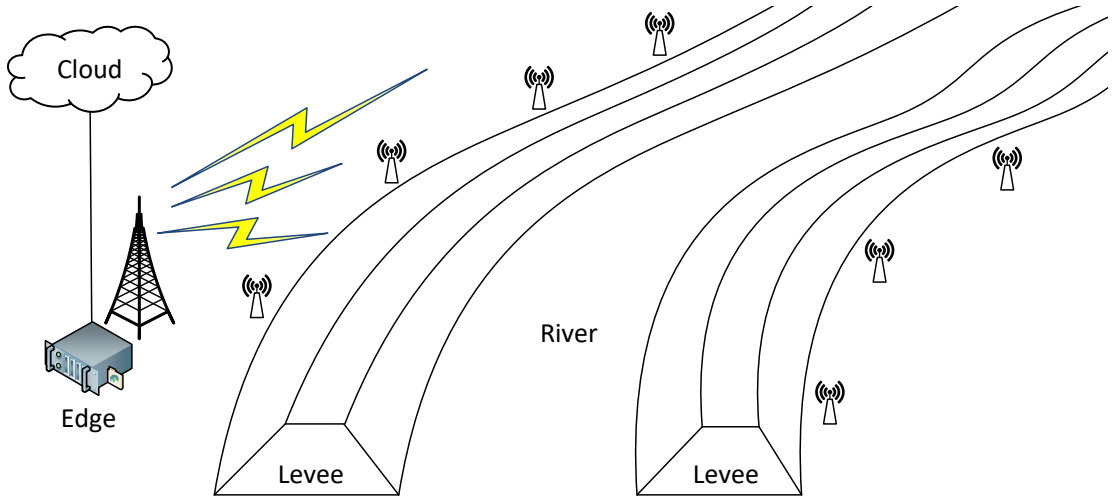


Figure 4.1: Levee monitoring system.

IoT devices are positioned along the river to monitor the physical aspects of flood barriers, such as temperature, humidity, and movement. The data collected by these sensors is initially processed at an edge station situated nearby. The processed data is subsequently transmitted to a cloud data center for in-depth analysis. The exchange of information between the devices and the edge server is facilitated through a wireless network.

The volume of data generated by sensors is influenced by the rate at which these devices perform measurements. Nonetheless, as the quantity of data transmitted grows, so

does the requirement for energy [150]. These devices are capable of self-powering as they come equipped with solar panels and rechargeable batteries.

For time-sensitive systems, it's crucial to transmit data regularly, especially during potential flood situations where real-time sensor information is essential, as noted by [151] in a holistic analysis. However, this frequent data transmission can exhaust the battery entirely, rendering the device non-operational until it can be recharged the following day by solar panels. In scenarios without an immediate risk of damage, the constant sending of sensor data can cause batteries to undergo deep discharge cycles. This not only leads to a swift decrease in battery life but is also not recommended due to its negative impact on battery health.

The research focuses on methods for adaptively managing data streams from sensors within the discussed system types. It explores two specific scenarios. In the first, the operation of the IoT system under standard conditions is examined, with the goal of maximizing battery life and minimizing full discharge cycles. The second scenario looks at the system's functionality during extraordinary circumstances, which necessitate continuous and frequent environmental monitoring and the transmission of sensor data.

4.4 Formal Model

We begin by presenting the required definition and system description to represent our research problem in Section 4.4.1. We formulate our problem using these definitions (Section 4.4.2). Table 5.3 lists all of the notations that were used in the chapter.

4.4.1 System Description and Definition

The infrastructure system X, which is represented as a quintuple $\langle O, PV, D, E, C \rangle$. O is a set of Osmotic Agents that respond to communication between the devices and is denoted by $O_o = \{id_o\}$, where id_o represent the identifier of the Osmotic Agents O_o. PV is a set of Photovoltaic panels located in each IoT device D_i and is denoted by $PV_p = \{id_p\}$, where id_p represent the identifier of the Photovoltaic panels PV_p. D is a set of IoT devices D_i and is denoted by $D_i = \{id_i, \delta_i, b_i, r_i, o_i\}$, id_i represents

the identifier of the IoT device D_i, δ_i represents the sensing rate of IoT device D_i, so, each IoT device observes its surroundings continuously over a given time interval, b_i represents the battery of IoT device D_i, r_i represents the renewable energy from the Photovoltaic PV panels, o_i represents the osmotic agent of the IoT device D_i. E is a set of edge devices E_e, each E_e is represented as $E_e = \{id_e, h_e\}$. Where id_e and h_e represent the identifier and the set of host machines h_e^1, h_e^2, ... for the edge device E_e, respectively. C is a set of cloud data centres C_c, and is denoted by $C_c = \{id_c, h_c\}$ where id_c is the identifier of the datacentre and h_c is the set of host machines h_c^1, h_c^2, ... for the cloud data center C_c, respectively.

An IoT application $\mathcal{A}_i$ is defined as a directed acyclic graph (DAG) of microservice $\mathcal{A}_i = \{\mathcal{A}_i^{\mu_1}, \mathcal{A}_i^{\mu_2}, ...\}$ in which each $\mathcal{A}_i^{\mu_j}$ represents a microservice to be executed. Each $\mathcal{A}_i^{\mu_j}$ has its own set of software (SW), hardware (HW), and quality of service ($\mathcal{Q}$) requirements. The combined requirements $\mathcal{R}(A_i^{\mu_j})$ for a microservices are shown in Equation (4.1).

$$\mathcal{R}(A_i^{\mu_j}) = SW^{\mu_j} + HW^{\mu_j} + \mathcal{Q}^{\mu_j} \tag{4.1}$$

In Equation (4.2), the total requirements of any application A_i is given by the sum up the requirements of all the microservices.

$$\mathcal{R}(A_i) = \sum_{\forall j} \mathcal{R}(A_i^{\mu_j}) \tag{4.2}$$

Data are generated by IoT devices D_i on a regular basis. The IoT device is treated as a passive entity, which means it does not handle data and instead sends it to the edge device. Each IoT device D_i have a battery b_i and a Photovoltaic panel PV_i that will recharge the IoT device D_i battery b_i continuously. The total battery capacity B_{total} is computed as given in Equation (4.3).

$$\mathcal{B}_{total} = b_{avl} + PV_{avl} \tag{4.3}$$

where b_{avl} is the IoT device D_i available battery capacity, and PV_{avl} is the IoT device D_i available Photovoltaic panel charging capacity. When the IoT device generates the

data from the surrounding environment and sends it to the edge datacenter E_e, that process will consume the battery. So, to calculate the overall battery consumption $\mathcal{BC}$ for each IoT device using Equation (4.4).

$$\mathcal{BC} = \frac{1}{s_r} \cdot t_r \tag{4.4}$$

where the s_r is sensing rate of the environment and t_r is the draining rate of sending the data to the edge datacenter E_e.

Table 4.1: Notations

Symbol	Description
X	The system infrastructure
O	A set of Osmotic Agents
PV	A set of Photovoltaic panels
D	set of IoT devices
E	A set of Edge devices
C	A set of Cloud data centers
h	A set of host machines
v	Virtual environment
δ	The data rate of IoT device
b	IoT device battery
r	The renewable energy from the Photovoltaic panels P
$\mathcal{A}$	An IoT application
SW	A software
HW	A hardware
$\mathcal{Q}$	Quality of Service
$\mathcal{R}$	Requirements
$\mathcal{B}_{total}$	The total battery capacity
b_{avl}	the IoT device D_i available battery capacity
P_{avl}	the IoT device D_i available Photovoltaic panel P charging capacity.
$\mathcal{BC}$	The overall battery consumption
s_r	Sensing rate of the environment
t_r	A draining rate of sending the data to the edge datacenter E
Q	Q function
A	An actions
S	A states
$\mathbb{R}$	A reward
α	Learning rate
γ	Discount factor
β	The weight

4.4.2 Problem Definition

Given an infrastructure $X = \{O, PV, D, E, C\}$ and a set of IoT applications $\mathcal{A} =$

$\{\mathcal{A}_1, \mathcal{A}_2, ...\}$, a suitable deployment solution Δ_m is defined as a mapping for $\mathcal{A}_i \in \mathcal{A}$ to X $(\Delta_m : \mathcal{A}_i \rightarrow X \forall \mathcal{A}_i)$ if and only if:

1. $\forall \mathcal{A}_i^{\mu_j} \in \mathcal{A}_i$, $\exists(\mathcal{A}_i^{\mu_j} \rightarrow v_h)$ where, $h \in \{h_e \cup h_c\}$
2. $\forall \mathcal{A}_i^{\mu_j} \in \mathcal{A}_i$, if $\mathcal{A}_i^{\mu_j} \rightarrow v_h$, then $HW^{\mu_j} \preceq v_h^{HW}$ and $SW^{\mu_j} \preceq v_h^{SW}$
3. $\sum_{\mu_j} HW^{\mu_j} \leq v_h^{HW}$ and $\sum_{\mu_j} SW^{\mu_j} \leq v_h^{SW}$

All the requirements to find a suitable deployment solution are considered in the definition given above. Requirement 1 indicates that a mapping between $\mathcal{A}_i^{\mu_j}$ and a virtual environment $v_h | h \in \{h_e \cup h_c\}$ must exist for every microservice belonging to the IoT application $\mathcal{A}_i$. Requirement 2 confirms that the hardware and software requirements of the microservice must be satisfied by v_h if a microservice $\mathcal{A}_i^{\mu_j}$ is deployed to a virtual environment v_h. Finally, requirement 3 limits the number of microservices a virtual environment can execute at any time t.

The primary goal of this study is to find the best solution for all applications $\mathcal{A}_i$ such that the overall battery consumption $\mathcal{BC}_{\mathcal{A}_i}$ is minimum. As given these requirements, we can represent the problem as shown below.

$$\begin{aligned} &\textbf{minimize } \mathcal{BC}_{\mathcal{A}_i} + \textbf{minimize } s_r^i \\ &\textbf{subject to:} \\ &\forall i \in \mathcal{A}_i, \forall j \in \mu_j \; \exists(\mathcal{A}_i^{\mu_j} \rightarrow v_h) \end{aligned} \tag{4.5}$$

The constraint states that all of the application's microservices $\mathcal{A}_i^{\mu_j}$ must be executed in a virtual environment (Equation (4.5)).

4.5 Osmotic Agents with RL

Modern IoT systems often employ a programming paradigm focused on data flows and stream processing [152]. These systems consist of computational processes that analyse data streams and transfer the results among themselves. They are implemented across various hardware platforms, including computational clouds, edge data centres, and the devices themselves. Osmotic computing [153], a concept that encapsulates this

methodology, offers flexibility by enabling these processes to shift between devices in a manner akin to the movement of solvent molecules through an osmotic membrane towards computational clouds or edge data centres. This adaptability allows the system to meet specific operational requirements, such as processing time or energy efficiency, based on the current operational context of the system.

In the proposed solution, we leverage the osmotic agents [138] concept. Each device has an agent associated with it that manages the device's resources.

In a traditional reinforcement learning approach, it is typically assumed that there is an interaction between an agent and an environment, with a critic evaluating the actions taken by the agent. However, the proposed solution deviates from this framework by introducing two environments: an internal environment, which acts as a virtual representation of the device, and an external environment, which represents the actual device.

Initially, the state of the internal environment is updated based on observations from the external device. Subsequently, the assessment of the actions taken on the external device is conducted by considering the state of the internal environment.

Furthermore, we assume that in the case of a network of IoT devices forming a sensor network, these devices are functionally similar and operate in a similar manner. Essentially, they are independent but share similar state distributions. This implies that the internal environment serves as a generic representation of an IoT device within the system, and the knowledge update process may involve observations from multiple devices.

In the solution depicted in Figure 4.2, agents from various devices can communicate, forming a multi-agent system. To accommodate the limited computing and memory resources of IoT devices, the adaptation logic of the device is managed by an agent operating in the edge datacenter.

The edge agent's internal environment can be enhanced by incorporating extra information from external sources. In this instance, we're referring to weather forecast data, including projected weather conditions and cloud cover for the current day and the following day.

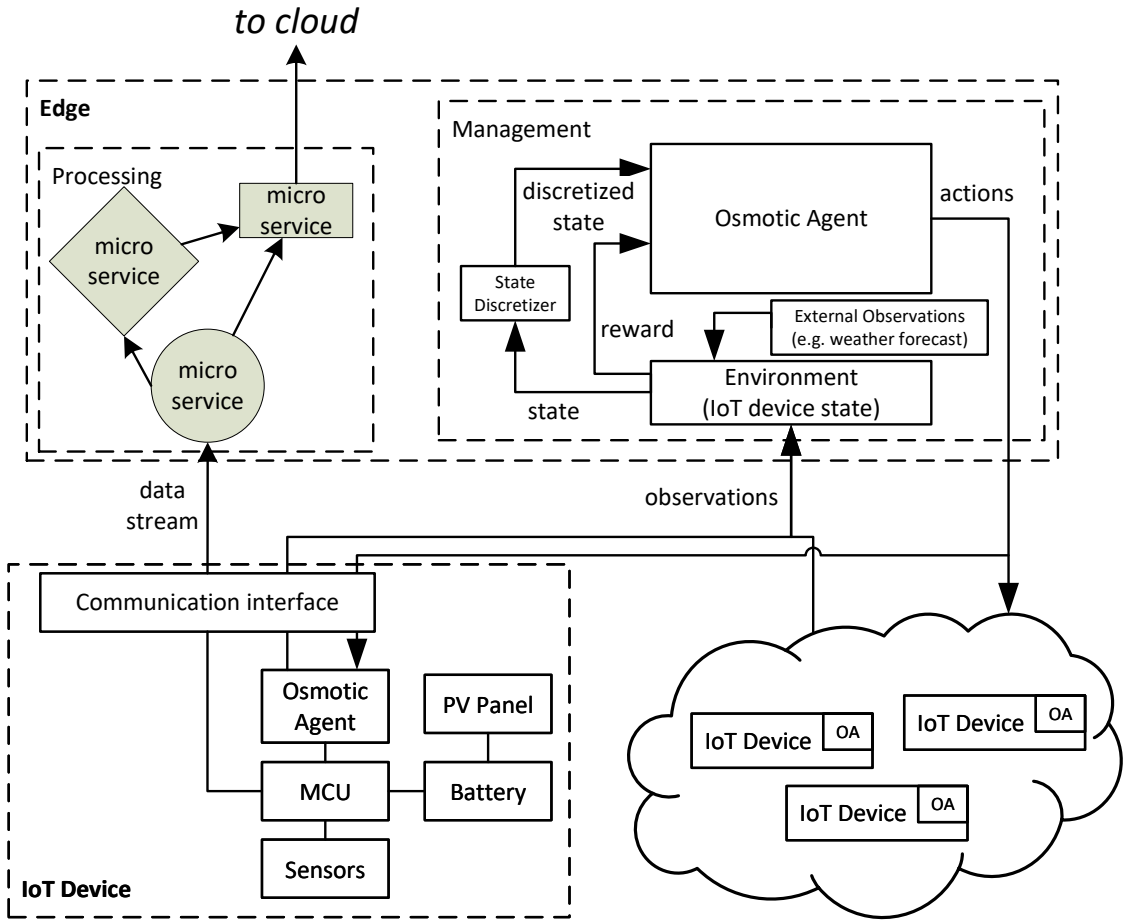


Figure 4.2: System architecture.

4.5.1 Q-Learning Algorithm

We chose to employ the traditional Q-Learning algorithm to develop the RL logic for the agents controlling the sensing rate. This algorithm, which is model-free, learns the value of an action taken in a specific state. In our approach, the available actions for the device are equivalent to selecting the sensing rate, i.e., $A = \{s_r\}$. Hence, the function Q is defined as:

$$Q : S \times A \rightarrow \mathbb{R} \tag{4.6}$$

Updating the value of the Q function is done using the Bellman function as an iterative update using the weighted average of the old and new values:

$$Q^{new}(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot [r_t + \gamma \cdot \underset{a}{argmax} Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4.7}$$

The α parameter is responsible for the learning rate, i.e., how much new values during learning affect updating the current values. The γ parameter is responsible for the discount factor, i.e., how important long-term rewards are compared to short-term ones. Parameter values influence the learning process and are application dependent. Typically $\alpha = 0.1$ and $\gamma = 0.8$ are assumed.

4.5.2 State Discretization

A discussion is needed regarding the number of potential environment states. Preliminary analysis suggests that DeepRL [154] methods can effectively solve RL problems with a large state space. However, in our solution, we chose to discretize the state space to reduce the resources required for algorithm training and implementation on real IoT devices. The possible state values for the problem at hand are listed in Table 4.2.

Table 4.2: Discretized states used in the RL algorithm.

Observation	**Number of States**	**State Discretization**
Today Forecast	3	*cloudy; partly cloudy; sunny*
Next Day Forecast	3	*cloudy; partly cloudy; sunny*
Month	3	$\{1,2,11,12\}\{3,4,9,10\}\{5,6,7,8\}$

4.5.3 Reward Function

The RL algorithm is based on the value of the reward obtained in response to the chosen actions. In our system, the reward function is a weighted average of two factors, where the β parameter determines the weight:

$$r_i = \begin{cases} \beta \cdot b_i + (1-\beta) \cdot \frac{min(s_r^i)}{s_r^i} & if\ b_i \geqslant 0.05 \\ 0 & if\ b_i < 0.05 \end{cases} \tag{4.8}$$

The first component concerns the device's battery level and assumes values in the range $[0; 1]$. The second component involves the sensing rate of the device. The more

often the device collects data, the higher the value is. The second component also takes values in the range $[0; 1]$.

In the case of the discussed systems, the amount of data transferred from the device is important, but the more important issue is to prevent the situation of a complete discharge of the batteries. Therefore, the reward function is 0 if the device has a critical battery level of less than 5%, and the β parameter was set to 0.2 to include battery level in the reward function.

We have set the critical battery level as 5% due to the possible inaccuracy of the battery capacity measurement and the potential need for a safe device's system shutdown. Therefore, we assumed that the RL agent receives a penalty if the battery level reaches the indicated value.

4.6 Evaluation

The solution was evaluated using the *IoT-SimOsmosis*[155] simulator, which was enhanced with a module for analysing renewable energy generated by photovoltaic panels. We assumed that the IoT devices monitor the temperature of the dyke, and their specifications are outlined in Table 5.5. The simulation utilised historical solar radiation data from 2016, sourced from the PVGIS database.

Table 4.3: IoT device specification used in the evaluation.

Device Type	Battery Capacity	Initial Energy	Battery Voltage	Solar Panel	Charging Current
Temperature Sensor	3000 mAh	2000 mAh	3.7 V	10 W	500 mA

We have experimented with different device management profiles for sensor data streams, which include both constant and adaptive profiles based on RL. The results of these experiments will be discussed in the following sections.

4.6.1 Constant Data Streams

In the case of constant management profiles, it was assumed that the device had a constant sensing rate value of 60 s, 90 s, 120 s, 150 s, 180 s and 210 s, respectively.

We observed changes in the device's battery levels during the experiment. The results grouped by months are presented in the Figure 4.3. We also counted during how many days in a year the device completely discharged the batteries. The results are presented in the Table 4.4. For measurements performed every 60 s, there were 166 days a year that the sensor stopped working due to a lack of energy, while the mean battery level was 47%. Most often, such situations occur during winter and spring seasons where solar radiation is lower than during summer periods. On the other hand, with measurements taken every 210 s, the device ran all year round without interruption having 89% of battery on average.

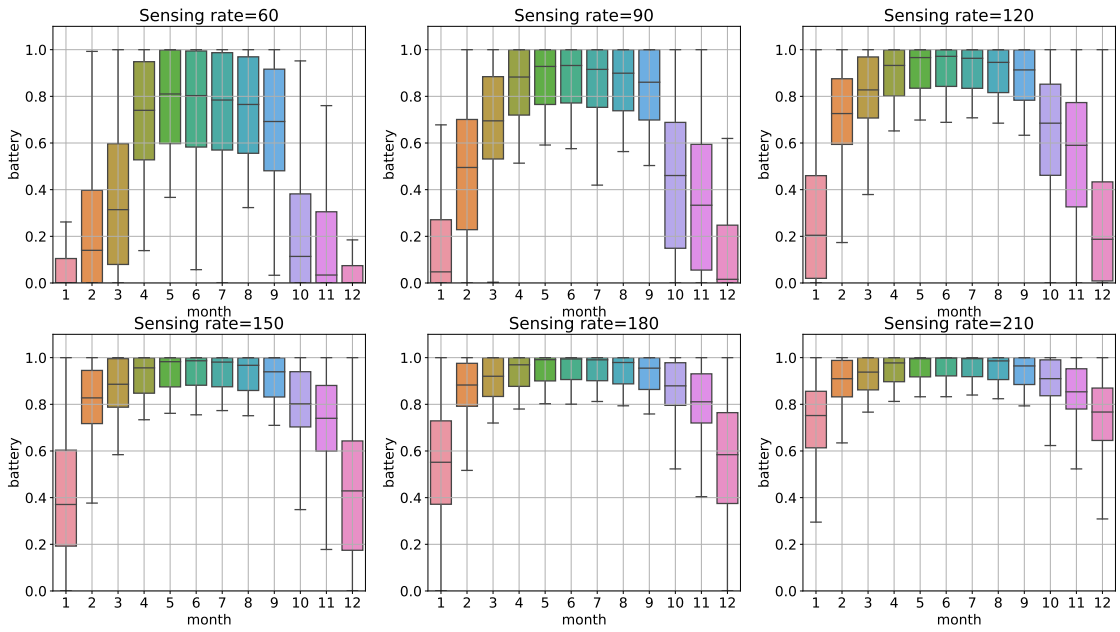


Figure 4.3: Battery levels of the device for various constant sensing rates. Colors of the boxes are related to the mean value of battery level.

One way to ensure frequent sensing is to increase the size of the PV panels and battery capacity. However, this approach is expensive and not very cost-effective. Instead, we suggest implementing dynamic data stream management for IoT devices to address this issue. The aim is to efficiently manage IoT devices to prevent them from stop working due to energy depletion and to deliver the necessary sensor data for flood predictions.

4.6.2 Dynamic Data Streams

When dealing with dynamic data streams, we expected the system to continuously learn which actions yield the highest rewards during operation. However, excessive exploration could lead to unexpected device behaviour. For instance, in the scenario depicted in Figure 4.4, random management actions such as altering the sensing rate can disrupt device operation. Consequently, there were 57 days when the device stopped functioning due to battery depletion.

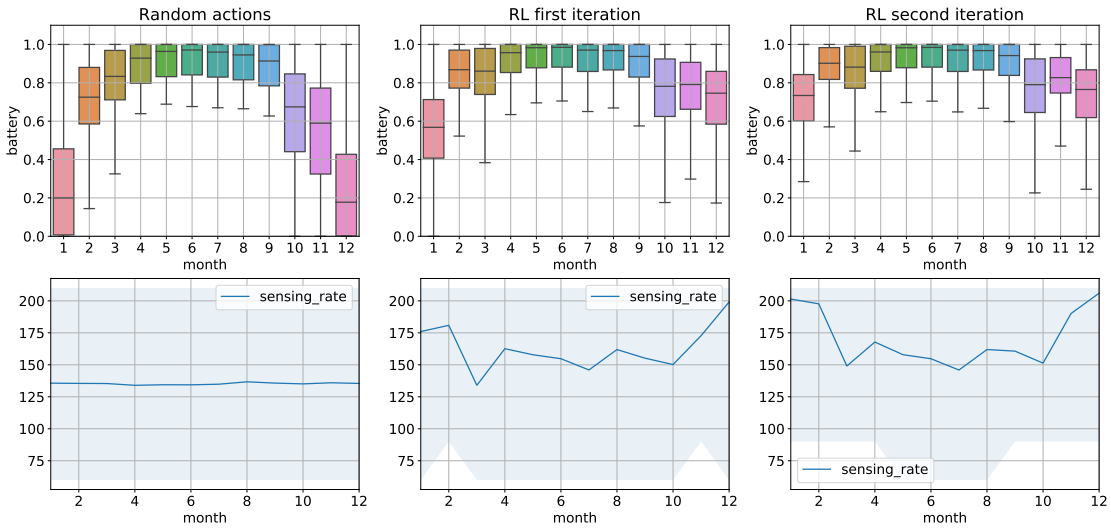


Figure 4.4: Battery levels and the selected sensing rates for the devices in RL based data stream management. Blue area represents *min* and *max* values of sensing rate, while the chart represent *mean* sensing rate value for the particular month.

Therefore, the exploration process was limited to a random generation of the Q table during system initialization. As a result, the initial adaptation actions taken by the device were random, which allowed for state exploration. It is especially visible in Figure 4.4 for the beginning months of the year (exploration resulting from a random Q table) where the battery was discharged. During the ending months of the year, the system has already developed an adaptation policy and thus preventing battery discharge.

The assessment persisted using the same historical data while the system continued its learning process. However, the actions taken only enhanced the previously employed policy. Consequently, the device operated continuously throughout the year, with an

Table 4.4: Data stream management profiles used in the evaluation.

Method	Mean Sensing Rate	Low Batt Days	Mean Batt Level
Constant 60 s	60 s	166	47%
Constant 90 s	90 s	105	62%
Constant 120 s	120 s	57	72%
Constant 150 s	150 s	26	79%
Constant 180 s	180 s	5	85%
Constant 210 s	210 s	0	89%
Random actions	135 s	57	72%
RL first iteration	162 s	8	83%
RL second iteration	170 s	0	85%

average sensing rate of 170 seconds. Throughout this period, the device batteries remained charged, and there was a 23% increase in the amount of data transmitted compared to the constant 210-second profile, as shown in Table 4.4.

4.7 Summary and Future Work

In our study, we introduce a model for dynamic data streams, which adjusts the rate of data transmission based on energy availability. This adjustment is made possible through the application of reinforcement learning, optimising energy that is expected to be generated by solar panels.

During our evaluation, we explored two distinct approaches to managing data flows from IoT devices. For uniform data streams, users determine their own settings. When minimising battery use is a priority, users can opt for the lowest possible data sensing frequency, meaning data is transmitted as infrequently as possible. Conversely, for those requiring comprehensive data on the environment being monitored, choosing to send data more frequently is an option, though this may lead to the IoT device's battery running out.

In the second case, the introduced dynamic data streams balance the operational

modes discussed. By utilising the Q-Learning algorithm, this method dynamically adjusts the rate of data transmission based on the availability of renewable energy sources. This approach ensures the efficient collection of data while also considering the longevity of the sensor's battery. The effectiveness of this solution was verified using past solar radiation data, demonstrating that it can enhance data transmission by up to 23% compared to other evaluated profiles, thus guaranteeing the device's uninterrupted functionality.

The operation modes of the discussed data streams, including the highest sensing rate, minimal battery usage, and the reinforcement learning RL-based dynamic mode, can be selected based on the user's needs and the specific goals of the IoT system.

Future work introduced could progress in two main directions. Firstly, there's the potential for the devices to collaborate directly, enhancing their proficiency in managing tasks as they accumulate experience and knowledge. In this scenario, RL agents could autonomously control individual devices, sharing information about their actions and, where applicable, the rewards received in particular system states. Secondly, the focus could shift to a thorough examination of the environment observed by the sensor network. This approach would allow for a strategic monitoring method where devices take turns recording data, rather than doing so all at once, based on an in-depth analysis of the surrounding environment.

5

Optimising data processing and handling misconfiguration policy failures in IoT systems using reinforcement learning

Contents

Summary

IoT and edge devices are still undergoing fast development, making resource optimisation and efficient failure management crucial. However, determining how to adapt and reconfigure IoT system faults and failures more effectively is still a considerable challenge. This chapter introduces a novel framework for an adaptive multi-agent system for fault detection and handling in an IoT system. We utilise MARL, where each agent observes and learns from the IoT device. Then, interaction among the various agents is developed to improve each agent's learning by taking into account the knowledge of the agents at the different devices in the network. The aforementioned method enables the detection of abnormalities throughout the whole system architecture. We also conducted an extensive evaluation in real environments to demonstrate the efficiency of the proposed method and show that our proposed method outperformed the existing techniques in detecting and handling misconfiguration faults, overcoming cascaded failures, and enhancing the data processing in nodes effectively.

5.1 Introduction

The IoT has rapidly emerged as a new paradigm for connecting devices for data collection, information sharing, and effective data processing. IoT faults can happen in various forms and conditions. Comprehending IoT faults aims to enhance IoT systems' stability and efficiency. It suggests the development of robust strategies and remedies to prevent, identify, and handle malfunctions. Various terms like failures, errors, anomalies, uncertainties, and defects are used interchangeably to denote IoT faults [156–159]. These faults may stem from hardware malfunctions, software glitches, environmental factors, network issues, or human errors. Moreover, in the IoT realm, faults can vary widely in severity, from minor glitches to major disruptions impacting the entire system. Furthermore, faults can be classified based on the nature of the flaw. For instance, it is essential to highlight the importance of the sensor's reliability, such as the calibration. Faults can adversely impact performance and dependability, according to the authors in [12].

Cascading failures pose a significant threat to IoT systems due to the interconnected

nature of various device functions and structures. These failures can trigger unanticipated and often undesirable state changes in other IoT devices, leading to serious cascading failures. Therefore, ensuring the robustness of an IoT system requires understanding the causes and physical mechanisms of cascading failures, considering their consequences in system analysis and modelling, and developing adaptability against such failures [160], [161]. Cascading failures occur when the failure of a single component initiates a series of failures in other components, causing severe damage to the entire system, as well as society and the environment. Such failures can occur in various applications, including IoT devices worn on the human body, transportation networks, and power grids.

A chain reaction of often unexpected and typically unwanted changes may be triggered in other IoT devices, leading to or speeding up significant chain reactions of failures. Thus, it's essential to ensure an IoT system's durability by comprehending the causes and physical processes behind these chain reactions of failures. This includes considering their impact in system analysis and modelling, and also developing resilience to such failures [160], [161]. Chain reactions of failures are particularly concerning when the malfunction of one component causes a domino effect of failures across other components, causing extensive damage to the entire system as well as to society and the environment. This scenario can occur in various settings, including IoT devices worn on the body, utilised in transportation networks, and integrated into power grids. [162], [163]. One IoT device experiencing overload may cause malfunction and cascade failures. Putting a device offline for an upgrade or maintenance might also cause cascading failures. Redistributing the workload of a malfunctioning device across other devices in the system might push these devices beyond their limitations, leading to their eventual overload. This overload spreads through the system due to the interconnected interactions among its various components, rather than through direct contact, which would be the result of structural dependencies [164].

Overall, this chapter tries to solve the research question:

How can we identify, detect, and handle faults in IoT and edge devices on the IEC continuum systems so the data quality is not compromised?

To address this research question, this chapter proposes a MARL-based adaptive sys-

tem which uses the collaboration of agents from their interactions in the shared environment to find the most flexible and effective fault-handling policy. As a result, MARL is appropriate for dynamic and complicated fault scenarios. The proposed method outperforms the static techniques with pre-established principles that are not capable of reacting to changing fault patterns, changing system circumstances, and the dynamic fault flow in the IoT system. To summarise, the main contributions of the chapter are shown as follows:

- We discussed and clarified faults and cascaded failures in the IoT system.
- We proposed a Multi-Agents Reinforcement Learning-based dynamic method to detect and handle faults in IoT systems.
- We evaluated our proposed approach's performance using actual IoT devices and real data.

The structure of the rest of the chapter is as follows: An overview of the related work is covered in Section 5.2. The fault modelling is covered in Section 5.3, while the system concept, system architecture, and the proposed concept based on reinforcement learning are covered in Section 5.4. The evaluation and results are presented in Section 5.5. In Section 5.6 the chapter is summarised, and suggestions for further research are made.

5.2 Related Work

Detecting and handling faults and failures using an adaptive method in IoT systems is widely described in the literature. [106] proposed a self-adaptive system with a technique for recovering resources in the event of node failure. The platform layer is responsible for managing the resources used for infrastructure to run applications. As a result, a Greedy Nominator Heuristic (GNH) was applied to the risk evaluation of resources in order to prevent allocating them to unreliable nodes and to guarantee higher service availability. An infrastructure that employs the replication strategy is used to guarantee this service. In other words, GNH has been verified with a real-

world smart city application that uses a centralised resource controller to monitor and manage data on surface water floods.

By identifying the optimal checkpoint interval values that impact the data recovery process in the original Apache Kafka pipeline, [165] contributes to enhancing fault tolerance within the structure of the Apache Kafka pipeline. Research demonstrates that employing the optimal checkpoint interval time in the design substantially reduces data loss.

IoTREPAIR [166] is an IoT fault management system that collaborates with fault identification modules to oversee devices experiencing issues. It comprises a set of fault-handling functions designed to tackle various types of faults. Additionally, it integrates a fault handler, based on this set, to autonomously manage faults associated with IoT. Observing the states of each sensor and how they relate to the states of their neighbours makes diagnoses that are made easier by developer configuration files and user preferences.

In a recent study, it was proposed that RV controllers use RL to recover from errors and attacks [167]. To identify representative faults and attacks from normal operation, this technique calls for training the policy with them. In practice, it can be challenging to find such typical faults and attacks. Furthermore, because of gyroscope attacks, the RL-based controller is unable to manage the altitude decline in both healthy and faulty conditions, which would ultimately lead to a crash.

A unique framework called MASAD is introduced in the study [168]. This approach focuses on detecting anomalies within microservices in industrial settings by effortlessly integrating a multi-agent system and RL techniques. It is designed to accurately pinpoint anomalies at both the local and broader system levels. each agent is customised through reinforcement learning, monitoring certain microservice behaviours, and exchanging findings. While implementing deep learning and pure data mining in specialised fields sometimes requires careful changes, MASAD is capable of identifying abnormalities in microservices data.

[169] discusses the escalating issues with the reliability of IoT systems, particularly those that are crucial to operations. Making sure of fault tolerance is essential due to

the complexity of the IoT, including the diversity of devices and the variety of communication protocols. The study provides fault injection, a method for purposefully causing system failures, to evaluate the robustness of these systems. They investigate how systems react to these purposeful failures by adding a fault-injection module to a well-known publish/subscribe broker, highlighting fault-injection's function in promoting fault tolerance in IoT environments.

The study [170] highlights the value of real-time fault detection for smart grids and the difficulties with deep learning in cloud-based systems due to internet transmission latency. The research presents a cloud-edge cooperative fault detection solution for smart grids that addresses these issues and is enhanced by lightweight neural networks on edge devices. The investigators suggest a deep reinforcement learning-based approach to optimise the allocation of communication and computation resources. This technique increases system throughput and communication effectiveness while improving solution speeds and ensuring timely data transfer. Although promising, the newly presented RL technique in this case provides a novel method of scheduling and allocating resources.

Therefore, our system effectively combines detecting, identifying, and handling faults and failures, as well as optimising the device throughput in the IoT system dynamically. It is thus sufficiently flexible to deal with faults and failures during data streaming processes. Table 5.1 summarises and compares the specific characteristics of the recently developed and our suggested systems.

Systems	**Features**								
	Fault Detection	**Fault Handling**	**Fault Identification**	**Dynamic Faults**	**Real Experiment**	**Reinforcement Learning**	**Edge processing**	**IoT devices**	**Allocating Resources**
Almurshed [106]		✓			✓		✓	✓	✓
Aung [165]		✓			✓				✓
Norris [166]	✓	✓	✓				✓	✓	✓
Fei [167]		✓				✓		✓	
Belhadi [168]	✓				✓			✓	✓
Duarte [169]	✓	✓	✓				✓	✓	
Li [170]	✓	✓				✓	✓		
Proposed	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.1: Comparing several related works to our proposed framework

5.3 Fault Modelling

It is crucial to comprehend the problem of misconfigured software policies, particularly in IoT and edge devices that activate the routing of sensor data. The aforementioned fault can occur due to several factors when data is transmitted between IoT components, such as sensors to edge nodes. Typically, the data pipeline starts from the sensing layer and ends at the processing layer (sensor data routing). The defect appears, potentially stemming from various components in the system, either hardware, software, or network components. However, the hardware and network components also can be significant sources of sensor data routing misconfiguration.

Fault Analysis	
Fault	**Improper data route**
Component	**Software**
Cause	**Misconfiguration**
Pitfall	**Overloading**
Description	**Sensors process data on single edge node**

Table 5.2: Detailed characteristics and facets of the fault.

To further elucidate, we are addressing a data routing problem that arose due to a fault in the policy implementation. This is being rectified to ensure the system operates dependably. An improper data route can place a substantial load on the edge layer, spiking CPU usage. Additionally, a thorough delineation of the characteristics and multifarious facets of the aforementioned fault is listed in Table 5.2.

5.3.1 Software Misconfiguration

The term software misconfiguration in the IoT application pertains to a status where the system is improperly configured. It is causing unpredictable behaviour. Various factors might induce it, including human error, software flaws, malfunctions, or external factors. The software component can greatly contribute to fault indicators within the system, including software bugs, stale updates, incorrect calibration, improper sampling rates, or incorrect data routing [171]. However, a misconfiguration is not necessarily to be raised as an initial erroneous in the system, but rather, the prior configuration of the system might no longer be adequate to the current system state and

thus might result in the system misconfiguration. Therefore, the previous state of the misconfiguration becomes invalid, and the system requires a reconfiguration to keep functioning properly. Accordingly, the main objective of this chapter concentrates on the latter pitfall, which is the need to reconfigure sensor data routing.

5.3.2 Cascaded faults

The potential consequences of a defect within an IoT system can range from negligible to substantial effects. The defect pertains to a misconfiguration of policies, specifically regarding imprecision in data routing. As depicted in Figure 5.1, in the event of misconfiguration within the path of transmitted data, the system may experience diverse errors. These errors may manifest as edge overloading, resource leakage, poor response, limited throughput, or data loss.

Such occurrences can significantly impede the system's efficiency and result in system failure. We encountered errors manifesting with the processing at the edge, likely caused by the sensor transmitting data in an incorrect direction. As a result, an excessive amount of data is being processed, leading to data loss and CPU overheating, ultimately resulting in system failure.

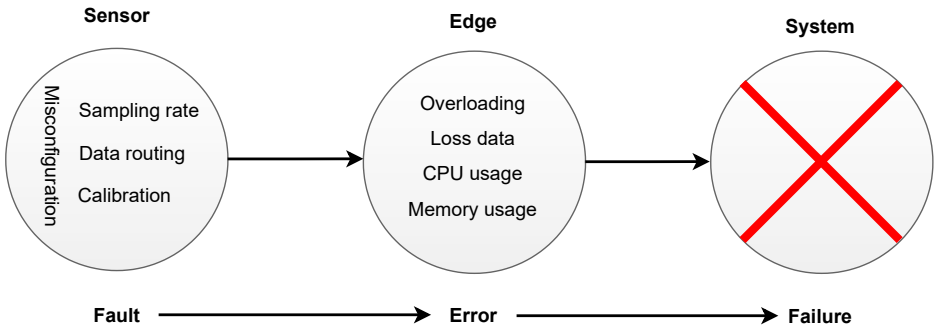


Figure 5.1: Improper calibration, incorrect sampling rate, or misdirected data routing between the sensor and edge layers can lead to errors like overloading, which may eventually cause a system failure. This sequence of fault-error-failure shows the cascading effect of faults in the system.

5.4 System concept

In this section, we discuss the concept of our system as shown in Figure 5.2. Our system has the ability to self-learn to know the point in time when it is necessary to monitor the system, detect faults, or reconfigure it while processing sensor data.

In this system, we have two types of agents deployed across all IoT and edge devices, and they communicate with each other to share information and collaborate. Agents on IoT devices manage the data, the processing in edge devices, and the failures by monitoring the state of the system, the changes that occur in the CPU of the edge devices, and the amount of processed and unprocessed messages that are waiting in the edge device processors queue. So, i.e., high CPU utilisation means an increasing number of elements in the queue. An unusual increase in the queue will result in CPU overload, causing the CPU to fail. The monitoring is done using Change Point Detection (CPD) technology, which will be explained later in Section 5.4.1. Then, heal the detected faults that may occur in the data routing or the edge device's CPU using MARL.

As for the edge device agents, their job is to collect the state of the device's CPU and the amount of processed and unprocessed data waiting in the edge CPU queue. Then send this data to the agents on the IoT devices to be analysed.

Moreover, in this system, clients work in two phases the detection phase and the handling phase. The state of the system is monitored by agents in the detection phase, and if it detects some quality degradation or decline in quality metrics, the system knows that something is wrong. Then the recovery phase will be activated by starting a reinforcement learning algorithm to heal the system. After a period of time and after returning the system state to normal, it switches back to the detection phase.

5.4.1 CPD

When the characteristics of the time series change, the challenge of identifying the unexpected shift in the data is known as CPD [172]. These discoveries, along with the

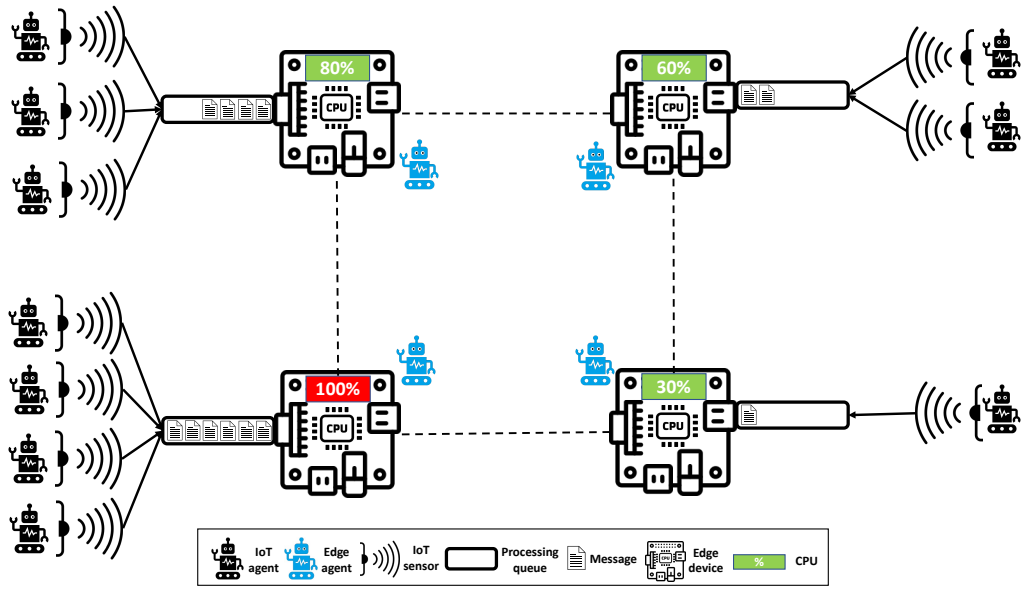


Figure 5.2: System concept

underlying structure, aid in the identification of time series change points in the event that a shift in system behaviour takes place [173].

The objective of the initial CPD research, which dates back to the 1950s [174, 175], was to identify a change in the meaning of Gaussian variables that were Independent and Identically Distributed (IID) for the intent of industrial quality control [176].

In other definition, the technique of locating times when a system's behaviour significantly changes is known as CPD. This change might be the result of the system's failure or an anomaly. Since a defect frequently results in a change in the result of the system, CPD is frequently used to detect problems in systems.

CPD techniques are divided into two primary categories: online techniques that seek to identify changes in the actual time immediately as they happen and offline techniques that identify changes after all data has been received. The first phase is often known as anomaly detection or event, whereas the last phase is occasionally referred to as handling or healing [177, 178].

So, we are using the CPD technique to detect faults that happened in the system, as described in the system concept shown in Figure 5.3. The rewards are monitored over time using this method. When a certain period of time passes with lower rewards, the system switches to the detection stage. The system tries to determine what caused

the reward to change. Then the system switches to the handling stage as soon as the cause is identified. During the phase of handling, the system makes some changes to the configuration of the system, trying to return the rewards to their initial value.

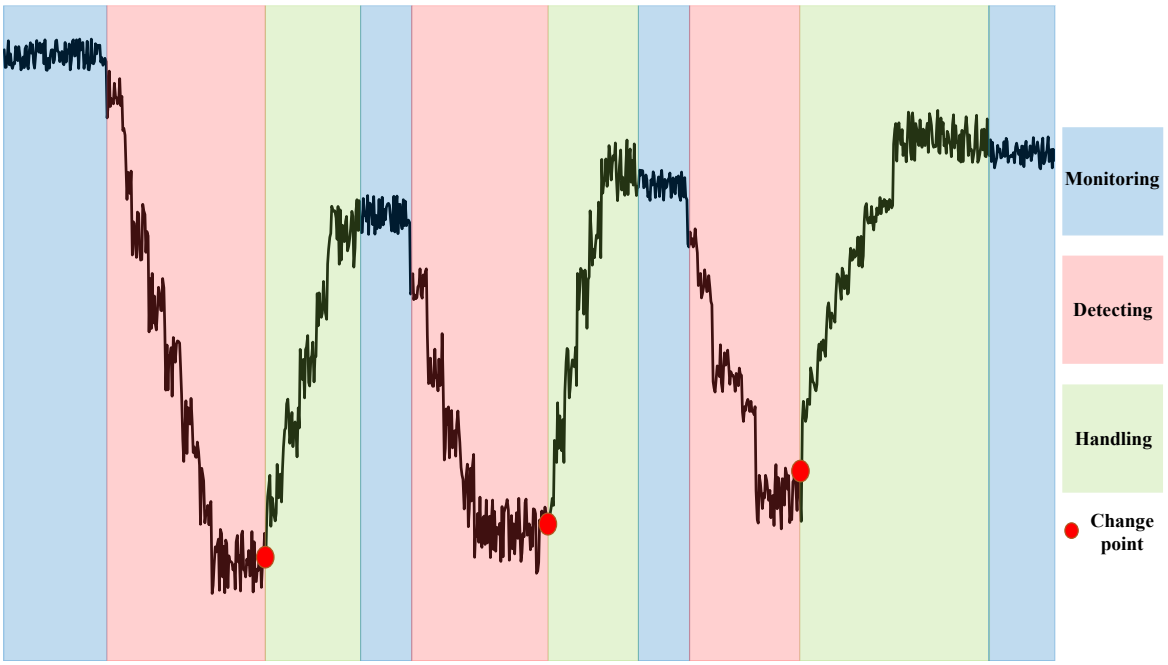


Figure 5.3: CPD technique

Rewards, faults, and CPD are related in the following ways:

- The system's performance is measured by the rewards. Any significant alteration in the system's output may result in the rewards to be changed.
- CPD is for detecting changes in the system that are caused by anomalies or failures.
- The output of the system could significantly alter as a result of failures.

Therefore, the system concept could potentially help ensure that rewards remain at a high level by applying CPD to find failures in systems.

Imagine a system in a smart home that keeps track of the functionality of numerous linked IoT devices, for example, smart lights, alarms, and cameras. Data about energy usage, processing speeds, and error logs are continuously collected by the system. The system goes to the detection phase if it notices a change point in any of those metrics, suggesting a possible malfunction or performance problem.

To determine the precise device or component generating the anomaly, the system examines the data gathered during the detection phase. Cross-referencing with other linked devices, checking through error logs, and comparing efficiency to historical data are some possible steps. The system shifts to the handling phase as soon as the primary cause is determined.

In this phase, the system handles the detected problem by implementing the necessary actions. The process may include notifying the user of impending dangers, remotely diagnosing the devices, arranging maintenance with the manufacturer, or automatically modifying device configurations. The main objective is to ensure that the smart home system is working smoothly and returning the device's performance back to normal.

So, the previous example shows how CPD could be used to successfully discover and fix problems proactively in an IoT system, guaranteeing the best possible user experience and performance.

5.4.2 System architecture

Figure 5.4 presents the system architecture. It is composed of the following components:

IoT devices: These devices are responsible for collecting data and sending it to edge devices. Action agents are also placed on the IoT devices to oversee deciding how to best allocate resources and deal with failures.

Edge devices: These devices are tasked with processing information received from IoT devices. To gather metrics from the edge devices and relay environmental conditions back to the IoT devices, monitoring agents are installed on the edge devices.

Collaboration channel: This network establishes connections between edge and IoT devices.

The collaboration channel would be used by the MARL agents to share information and work cooperatively. They might then coordinate their actions and communicate information. Moreover, in our system, we have two types of agents, i.e., an action agent and a monitor agent.

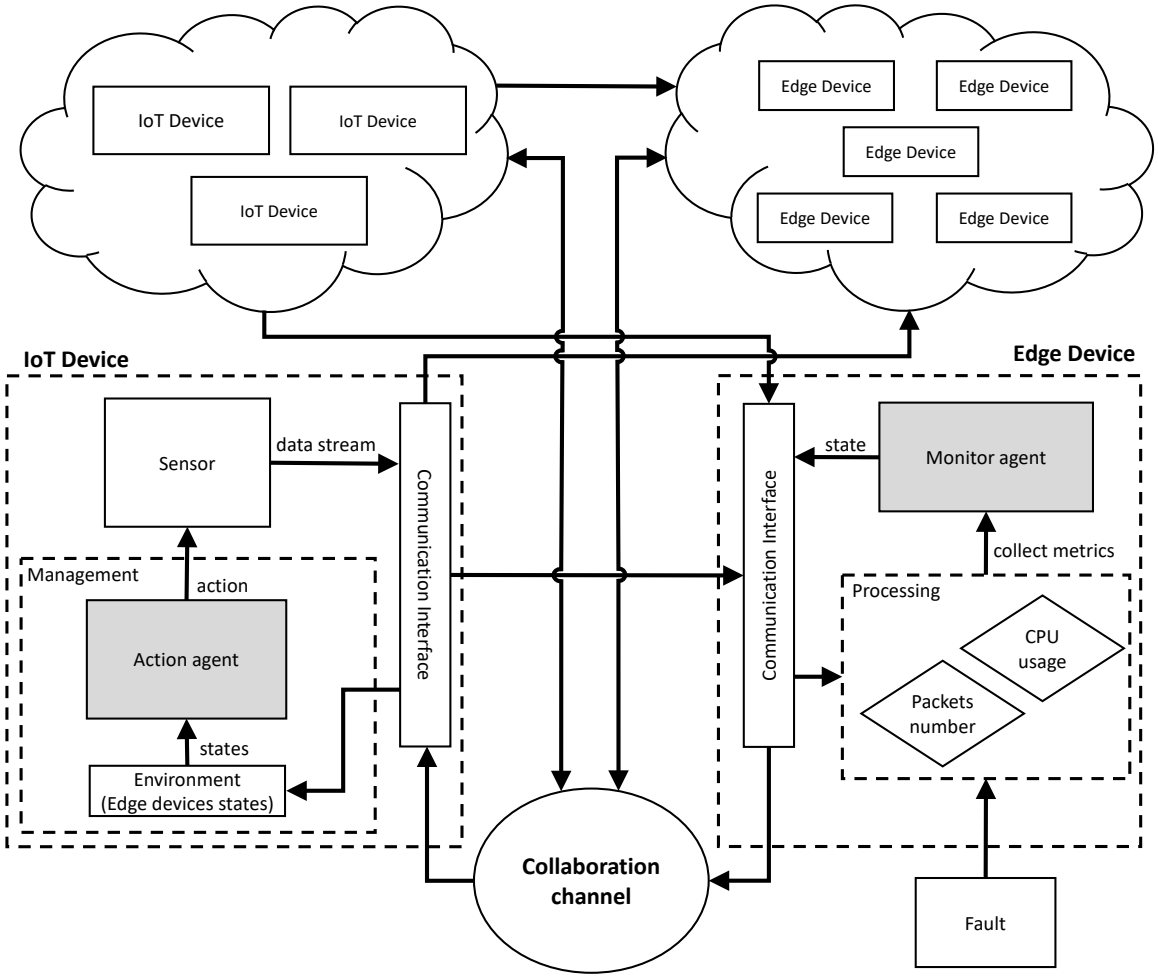


Figure 5.4: System architecture

An **action agent** on an IoT device, for instance, may notify other action agents of a failure so they can respond appropriately.

On the other hand, metrics like the number of packets and CPU utilisation are gathered by the edge device's **monitor agent**. The action agent on the IoT device would then get this information. After analysing this data, the action agent would decide how to distribute resources and deal with faults.

By monitoring the rewards and penalties it receives, the action agent would be able to detect any failures in the system. The action agent would need to act if it began to receive several penalties, which would indicate that something was wrong.

The action agent may choose to handle fewer packets overall or handle more packets. To lessen the consequences of a failure, the action agent may also choose to redirect the data flow to another edge device.

5.4.3 Reinforcement learning algorithm

In our approach, we implemented a model-free, off-policy RL technique known as Q-learning. This algorithm estimates the value of executing a particular action within a given state. The RL agents employ the Q-learning method to acquire ideal strategies for managing faults and allocating resources within the IoT framework [179]. Table 5.3 lists all the notations used in this chapter.

Table 5.3: Notations

Symbol	Description
Q	Q function
a	An action
s	A state
r	A reward
α	Learning rate
γ	Discount factor
a'	The action taken in the next state
s'	The next state
$\mathcal{SR}$	The *speedReward*
$\mathcal{UR}$	The *utilReward*
σ	The standard deviation
$state\mathcal{CR}$	The sensor count
$state\mathcal{R}$	The state rewards
QC	The queue count
$\mathcal{N}$	Node id
$\mathcal{F}$	Fault type
$\mathcal{T}$	The duration of the fault
$\mathcal{W}$	The workload

The Q-values (Q_r) are updated for each RL agent r $\in$ R using the Bellman equation as a basic concept. To iteratively update the Q-values, it takes the present reward, the highest anticipated future reward, and the learning rate, according to the following equation:

$$Q_r(s,a) \leftarrow (1-\alpha) \cdot Q_r(s,a) + \alpha \cdot (r + \gamma \cdot \max a' Q_r(s',a')) \tag{5.1}$$

where:

- Q_r(s, a) refers to the Q-value of agent r in state s and performing action a.
- r refers to the immediate reward received from the environment.
- α refers to the learning rate, which controls the weight of the new information $(0 < \alpha \leq 1)$.
- a' refers to the action taken in the next state in accordance with the existing policy.
- s' refers to the next state.
- γ refers to the discount factor, which specifies the significance of future rewards $(0 < \gamma \leq 1)$.

Moreover, the balance between the weights assigned to the new information and the current information stored in the Q-values is determined by the α parameter. It regulates how frequently the Q-values are modified in response to the RL agent's new experiences gained. The γ parameter illustrates how future benefits are prioritised over present gains. It establishes how much consideration is given to potential benefits while making decisions. Therefore, we assumed the $\alpha = 0.1$ and the $\gamma = 0.9$.

5.4.4 Reward Function

The value of the reward r received in response to the chosen actions is the basis of the RL algorithm. The reward function in our system is a weighted average of four factors:

$$r_i = \begin{cases} \frac{\mathcal{SR}+\mathcal{UR}+state\mathcal{R}+state\mathcal{CR}}{n} & if\ state\mathcal{CR} > 0 \\ 0 & if\ state\mathcal{CR} = 0 \end{cases} \tag{5.2}$$

where $\mathcal{SR}$ refers to the *speedReward* which is the reward of the sensing rate of the IoT device in the range $[0; 1]$. $\mathcal{UR}$ refers to *utilReward*, which is the reward of CPU utilisation of all edge devices. To equalise the CPU usage across all edge devices, we employed the standard deviation σ using the following formula:

States	CPU Utilisation(%)	Queue Count	Rewards
State0	0 – 10	QC > 0	0.1
State1	11 – 20	QC > 0	0.2
State2	21 – 30	QC > 0	0.5
State3	31 – 40	QC > 0	0.6
State4	41 – 50	QC > 0	0.7
State5	51 – 60	QC > 0	0.8
State6	61 – 70	QC > 0	0.8
State7	71 – 80	QC > 0	0.9
State8	81 – 90	QC > 0	0.9
State9	91 – 100	QC > 0	0.8
faultState	100 – 100	QC = 0	0

Table 5.4: IoT devices specifications used in the evaluation.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}} \tag{5.3}$$

Next, *state*$\mathcal{CR}$, refers to the sensor count, i.e., how many sensors send their data to a specific edge device. So, in order to create a load balance amongst the devices to manage the failure, our goal is to equally distribute the sensors across the edge devices.

Next, *state*$\mathcal{R}$, refers to the state rewards. We assumed that there are eleven states, and each state refers to a range of CPU utilisation. For instance, *state0* means that the CPU utilisation is between 0% and 10% and the reward is 0.1. The other states work similarly, except for *faultState*, where the CPU utilisation is 100% constantly and *queueCount QC* is zero, where the *QC* refers to the number of data in the edge device processing queue. So, when there are too many messages received from IoT devices, the queue will be excessively long, causing the CPU to be overloaded, which leads to delaying message processing and increasing the CPU temperature. Then it eventually causes the CPU to freeze, resulting in messages not being processed. So the reward function will be 0 for *faultState*. As shown in Table 5.4.

This means there is a fault that occurred in one or more edge devices. The final average reward r, is calculated by dividing the total rewards by four. So, in our case, the amount of data sent from the sensors and processed by the edge devices is vital, but it is more crucial to avoid the edge device's CPU utilisation being completely occupied and no data being processed.

5.4.5 Actions

Along the learning process, the agents perform actions, obtain rewards according to their states and actions, and alter their policies to optimise their overall rewards over time. We created appropriate actions in our solution to deal with the failures the system found and optimised the data processing. Every agent has four actions:

- **Increase rate:** When data is processed quickly, the sensor will increase the rate at which it senses data, i.e., sending more data to the edge device.
- **Decrease rate:** If the sensor detects that the data is being processed slowly, it will reduce the data sensing rate or transmit less data to the edge device.
- **Do nothing:** When an agent receives a high reward, it may indicate that the prior action was successful and there is no need to modify it for a while. In this case, the agent should do nothing.
- **Change edge:** Where the sensor sends the data to a different edge device. This action should be conducted in particular circumstances, such as when the edge queue is excessively busy or there is a failure event, to let the edge device recover.

5.5 Evaluation

We are using MARL in this experiment to make our IoT system adaptively recover from faults successfully, therefore, we can accomplish the main goal of this chapter. Using a set of Raspberry Pis as an IoT device in the experiment to guarantee the proposed approach's feasibility on real devices. The experiment precisely imitates actual operational scenarios by utilising physical IoT devices, such as sensors and edge devices. Evaluating this solution on real devices offers a more precise assessment of how well it handles cascading faults and optimises data processing.

5.5.1 Setup

In the experiment, we used eight Raspberry Pi's, three of them as IoT sensors and five as edge devices. A router to use as a collaboration channel. The component

specifications that we used in the experiment are shown in Table 5.5. We have obtained a real sensor data set from Urban Observation at Newcastle University[139]. This dataset consists of samples collected from three gases, i.e., CO, NO, and PM2.5. We used the MQTT protocol to send and receive the data between the devices.

Device model	**Processor**	**Memory**	**Quantity**
Raspberry Pi 4 Model B	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	4GB	6
Raspberry Pi 3 Model B+	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz	1GB	3

Table 5.5: The specifications of IoT devices used in the evaluation [1].

Figure 5.5, shows the system data flow. We assumed that the sensors were collecting and sending the data to the edge devices to be processed. At the same time, the RL action agents that are implemented in the sensors observe the edge device's CPU utilisation and the number of packets in the processing queue and calculate the reward by communicating with the monitor agents that are implemented in the edge devices. Then the action agent will try to detect any faults by observing the rewards and penalties and handling them by performing specific actions and analysing the states and rewards. All Raspberry Pi's use the MQTT protocol to communicate via the collaboration channel.

The collaboration channels purpose is to facilitate communications between all devices. The RL agents will collaborate by sharing the state of the environment. There were two tests, the first without implementing the RL agents in the sensors. The second one involves implementing the RL agents in the sensors and edge devices. We used the same initial configuration file in both tests.

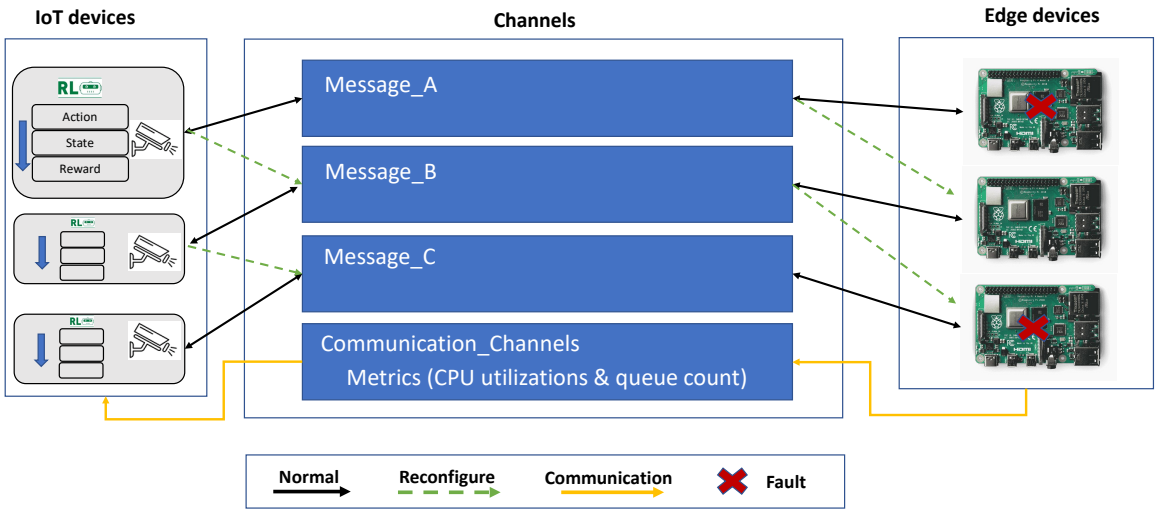


Figure 5.5: System flow in the experiment

5.5.2 *Fault Injecting Component*

Chaos engineering technique has been used to intentionally inject faults into the system to evaluate the reliability of computer system hardware or software [180]. Due to the fact that software fault injection only calls for modifications at the software state level, it is less expensive than hardware fault injection. Therefore, it is frequently simple to use software fault injection to test and evaluate the higher-level mechanisms of software systems.

Accordingly, we introduce a CPU fault injection component that injects faults into the edge nodes. It is possible for the CPU to run out of resources due to a high volume of activities. Application failures, such as slow response times and low throughput, may be the result of this.

The pseudo-code of the fault injection execution controller is shown in Algorithm 5. The edge node in which the fault will be injected ($\mathcal{N}$), the fault $\mathcal{F}$ (CPU), the duration of the fault ($\mathcal{T}$), and the workload ($\mathcal{W}$) are the inputs of Algorithm 5. This algorithm first generates a workload based on the fault (Algorithm 5, Line 5). Then, the *Injection* procedure is performed to assign the fault for injection duration ($\mathcal{T}$) (Algorithm 5, Line 7). Finally, the fault is injected into the selected node (Algorithm 5, Line 9).

Algorithm 5: Fault injection execution controller.

Input: $\mathcal{N}$ - node id,
N_l - list of nodes,
$\mathcal{F}$ - fault type,
F_l - list of fault types,
$\mathcal{T}$ - injection duration,
$\mathcal{W}$ - workload.

```
// Start the fault injection process
for each 𝒩 in N_l do
    for each ℱ in F_l do
        // Generate 𝒲
        GenerateWorkload(W)
        //Run Injection method
        inj ← Assing (ℱ, 𝒯)
        //Inject into the 𝒩
        𝒩 ← Inject (inj)
    end
end
```

5.5.3 Experiment results

The results of the experiment directly tackle the research problem of preventing cascading failures brought on by incorrect configuration in the system, especially the sensor data routing fault.

In the beginning, we ran the experiment with three tests. First, the **baseline**, which means no fault was injected and no action was implemented except the *Do nothing* action, i.e., the agents will not perform any action such as increasing or decreasing the sensor rate or changing the edge. The sensor rate was set to 3 seconds, i.e., the sensor sends data to edge devices every 3 seconds. The reason for choosing this rate is that, after we did several initial tests, we assumed that at this rate, edge devices can behave normally. Therefore, the goal of the first test is that we need to measure the performance of the normal configuration by calculating the average rewards. In the **second** test, we started to inject the fault into the edge devices, and we set the sensor rate to 3 seconds but took no action except *Do nothing*, to check how the system would behave. For the **third** test, we initiated the sensor rate with 3 seconds, injected the faults into the devices, and then ran the RL to start handling the faults in the system, as shown in Table 5.6.

Further, the sensors were configured randomly, i.e., with randomly located edge devices to send the data to be processed. In Figure 5.6 the results of the first test (the baseline)

Table 5.6: The configuration of IoT devices used in the evaluation.

Tests	**Sensor rate(s)**	**Action**	**Fault**
Baseline	Constant 3 s	No	No
Second	Constant 3 s	No	Yes
Third	Variable 3 s	Yes	Yes

were provided. We can notice from the start to the end that the average reward is stable between 0.4 and 0.8, which means that the system is working normally and there is no drop in the reward.

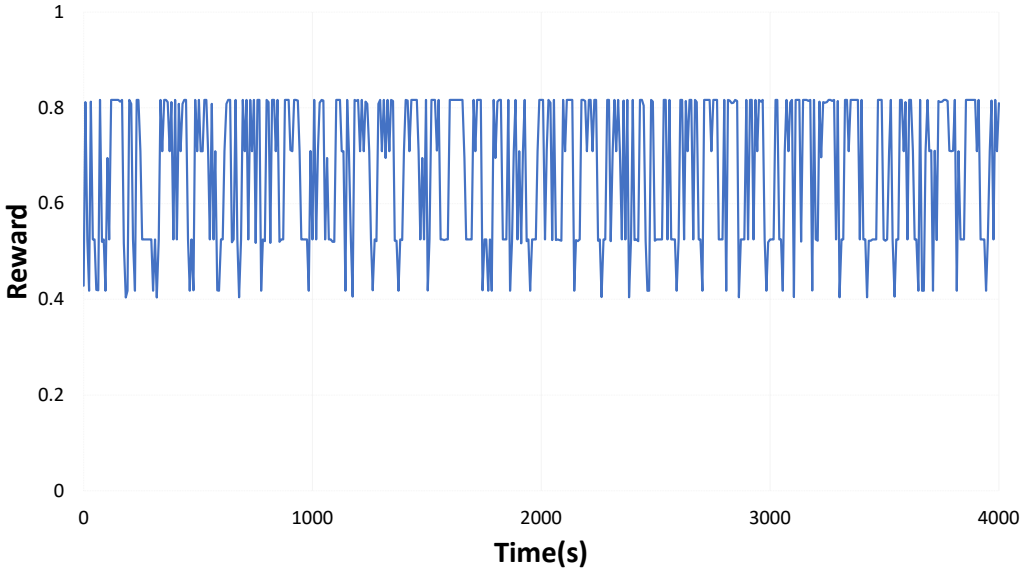


Figure 5.6: The average rewards of Baseline test

Next, Figure 5.7 shows the result of the second test. We can notice at the start that the average reward is between 0.6 and 0.8 for a while, which is a good reward. Then the reward drops to 0.0, which means there was a fault in the system, and unstable rewards continue until the end of the test. So, this leads to low throughput and delayed response times for the system.

Next, Figure 5.8 shows the result of the third test.

We suppose that when the system is running, it should learn online which actions result in the highest rewards. In the beginning, the average rewards are stable, with a good range. Then, the rewards started to drop to zero several times. As we mentioned in the system concept section, we are using the CPD technique to detect the faults that happen in the system. Thus, by monitoring the changing point in the rewards,

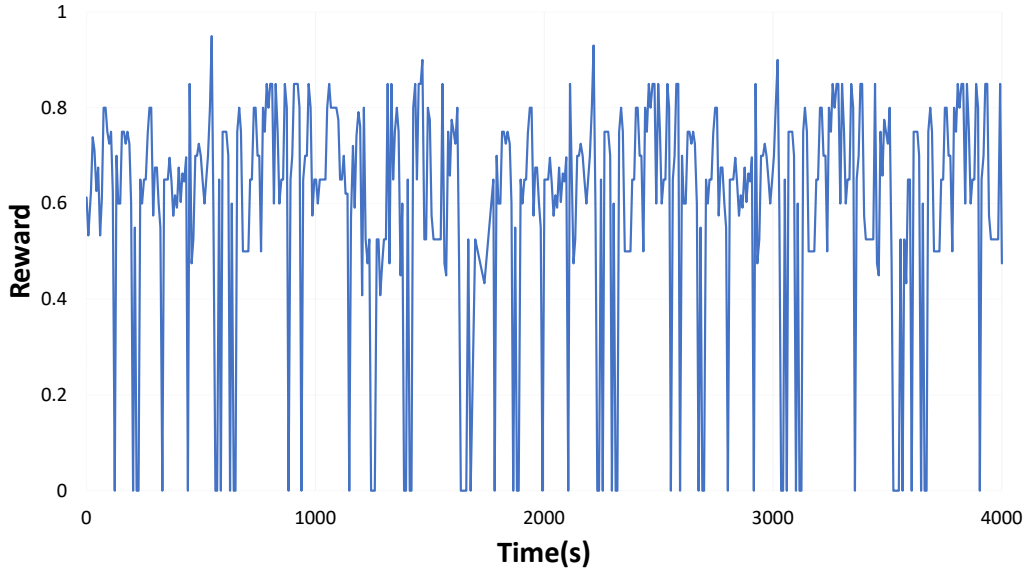


Figure 5.7: The average rewards of the IoT devices sending data to the edge devices without RL agents handling the faults

we can notice there are two points where the agents detect some faults that occurred, and then the agents will start handling those faults by analysing the metrics and performing different actions.

So, agents finally figured out the best actions they could take to overcome the faults and balance the rewards.

Finally, one of our goals in this chapter is to improve the performance of the data processing of the messages sent by IoT devices to edge devices. Therefore, we calculated the total number of messages processed in each test and compared them so that we could see the improvement in processing performance, as shown in Figure 5.9. Therefore, the improvement percentage in processing performance compared between the traditional system and our proposed solution is 23%.

The experiments show a good performance of identifying faults, reducing fault propagation, handling failures, and optimising data processing of our proposed solution.

5.6 Summary and Future Work

A novel framework for detection, identification, handling faults and failures, and optimising the data processing performance dynamically based on a multi-agent system

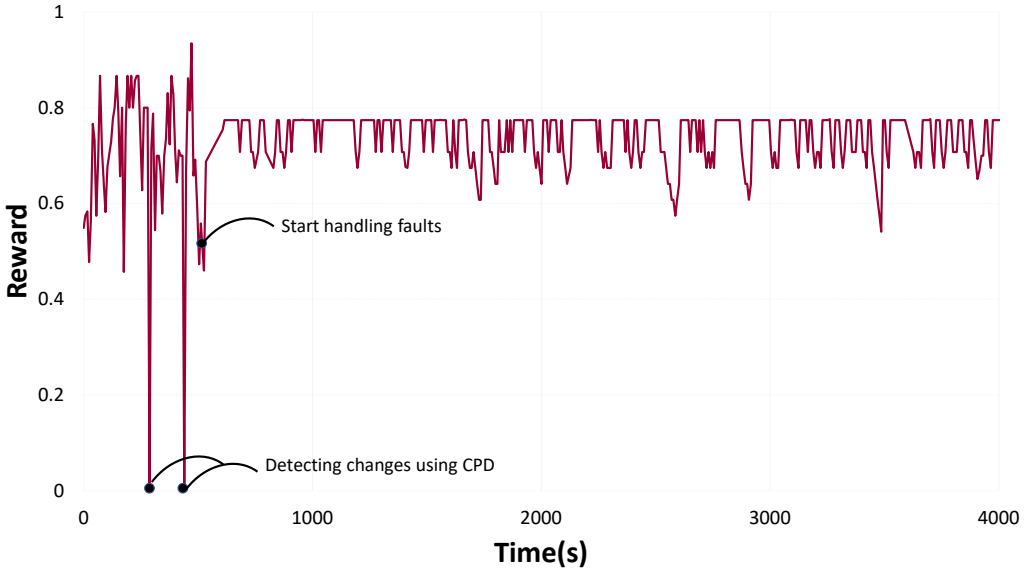


Figure 5.8: The average rewards of the IoT devices sending data to the edge devices with RL agents handling the faults

in an IoT system is designed. This system is developed using a multi-agent reinforcement learning algorithm, where each agent receives and observes data from the supplied node. Then, by incorporating the data of the agents of the various nodes in the network, the interaction between the various agents is formed to enhance each agent's learning. We have identified and explained what are the symptoms and causes of faults, especially misconfiguration data routing faults, and how they're related to cascaded failures. Using that information, our system can easily detect and handle anomalies or faults in IoT systems. To show the effectiveness of the proposed framework, we carried out an experimental investigation in real environments. Results showed that our proposed framework is more successful than the conventional methods at effectively detecting and addressing misconfiguration faults, overcoming cascaded failures, and improving the data processing in nodes. In future work, we can further extend this framework by applying more complex data routing, such as adding the cloud environment. Also, we can focus on the data itself, and ensure that there are no faults in it, to maintain the quality of the data.

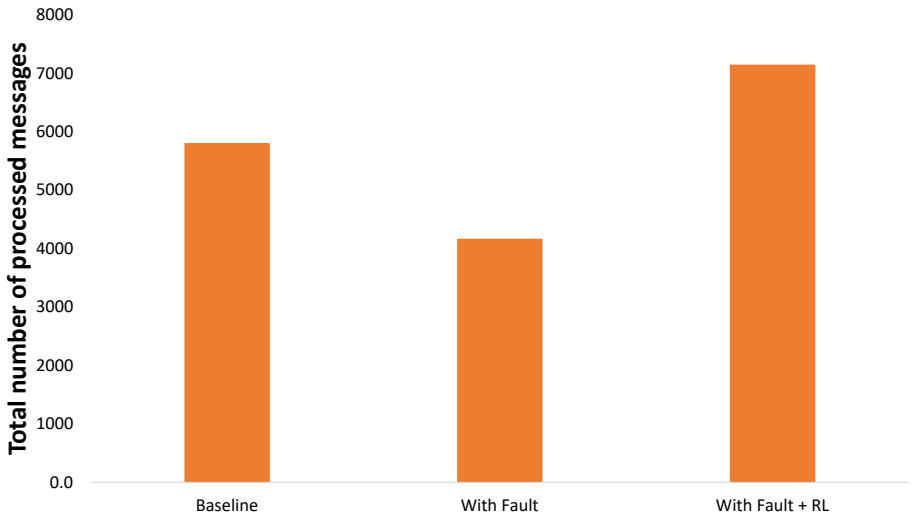


Figure 5.9: A comparison of the total number of messages processed between tests

6

Conclusion

Contents

Summary

In this chapter, we provide a summary of the research conducted in this thesis. We summarise the key findings and contributions and identify research challenges in the domain that could serve as directions for future works.

6.1 Thesis Summary

This thesis delves into the optimisation of time-critical data processing within the IEC continuum, highlighting the significance of low latency, device sustainability, and data quality for applications demanding real-time decision-making, such as infrastructure management, healthcare, and flood monitoring. Due to the limited computational power, limited bandwidth, low energy, and data integrity and reliability of energy-aware IoT devices and edge devices, optimising time-critical data processing is challenging as it requires an adaptive approach to efficiently tackle those limitations of the devices in the IEC.

In this thesis, we investigated various challenges associated with the optimisation of time-critical data processing within the IEC continuum and put forward solutions to ease the optimisation process. Specifically, this thesis presents the following contributions:

Chapter 2 provides an overview of the IEC continuum and IoT, edge, and cloud environments. Then it discusses the network management techniques in IEC. Next, it discusses the adaptive techniques for fault management and data quality in the IEC. It then discusses the challenges in time-critical data processing applications and how adaptive solutions are important to handle those challenges. At the end, we discuss the relevant literature reviews and identify the research gaps.

Chapter 3 We introduce a novel distributed traffic scheduling system that prioritises QoS based on multiple levels of queuing. This system aims to maintain high system throughput while reducing queuing delay and ensuring better QoS for latency-sensitive applications. Our scheduling system utilises multi-level queues to allocate network bandwidth, dividing it based on the latency sensitivity of incoming traffic. Addition-

ally, the bandwidth allocation in our system is dynamically adjusted by analysing network utilisation in real time. Through these methodologies, our system significantly improves latency, throughput, and energy consumption in edge-cloud environments compared to existing methods such as FCFS and SJF.

Chapter 4 The concept of dynamic data streams was introduced, which involves adjusting the data transfer rate based on the availability of energy resources. Utilising RL, the system can adapt to the anticipated energy output from renewable sources like photovoltaic panels. The implementation employs the Q-Learning algorithm to regulate the data transport rate, considering the level of renewable energy resources accessible, while also accounting for sensor battery life. This dynamic approach represents a compromise between various operational modes. The effectiveness of the proposed solution was assessed using historical data on solar radiation levels, demonstrating an increase in transmitted data compared to other evaluated profiles. This enhancement ensures the continuous functionality of the device.

Chapter 5 introduces a new framework designed to detect, identify, manage faults and failures, and optimise data processing performance dynamically within an IoT and edge system. The framework relies on a multi-agent system developed using a multi-agent reinforcement learning algorithm. Each agent within the system receives and monitors data from its assigned node. By integrating data from agents across the network, interactions between agents are established to enhance individual learning. We have outlined the symptoms and causes of faults, particularly focusing on misconfiguration data routing faults and their association with cascaded failures. To demonstrate the efficacy of the proposed framework, real-world experiments were conducted. The results indicate that our framework outperforms baseline methods in effectively detecting and resolving misconfiguration faults, mitigating cascaded failures, and enhancing node data processing.

6.2 Limitations and Future Research Directions

We offer inspiration for some directions of future research that the work in this PhD thesis can lead to.

6.2.1 Enhancing the Bandwidth Allocation

In Chapter 3, one of the limitations is its dependence on static bandwidth allocation, which may not optimise resource usage effectively under varying network conditions. Although the system features a heuristic semi-auto-adaptation algorithm for dynamically adjusting bandwidth slicing, it initially relies on fixed static percentages. This reliance can cause inefficiencies before the dynamic adjustments are implemented. To address this limitation, future research could explore the development of a completely dynamic bandwidth-slicing method that removes the need for static configurations. Such a method could utilise real-time data analytics and machine learning techniques to anticipate traffic patterns and allocate bandwidth. By continuously observing network conditions and adapting in real-time, the system could improve its responsiveness, resulting in better latency, and throughput, maximising resource utilisation, and overall network performance, particularly in more complex and large-scale IoT environments.

6.2.2 Enhanced Multi-Agent Cooperation for Dynamic IoT Networks

In Chapter 4, one of the limitations is the emphasis on individual IoT devices functioning independently, without taking advantage of the collaborative potential between devices. This approach limits the system's ability to optimise device management by utilising shared experiences and collective knowledge. In this chapter, we explored the potential for device collaboration in which RL agents function autonomously on individual IoT devices. These agents exchange messages containing actions and rewards corresponding to specific environmental states. As a future direction, the development of the discussed approaches can take two main paths. Firstly, there is the potential for devices to cooperate among themselves, thereby enhancing their experience and knowledge in device management. In this scenario, RL agents would operate independently on each device, exchanging messages detailing the actions taken in specific system states, along with the achieved rewards. Secondly, there is the possibility of domain analysis within the monitored environment where the devices, forming the sensor network, are situated. This would allow for selective monitoring of the environment,

enabling devices to take measurements alternately rather than simultaneously.

6.2.3 Advanced Data Quality Assurance Mechanisms

In Chapter 5, one of the limitations is that it primarily focuses on fault detection, misconfiguration handling, and optimising data processing performance, without adequately addressing the quality of the data being processed. This oversight can compromise the reliability of IoT systems, as the accuracy of data analysis and decision-making heavily depends on the integrity of the data collected. To address this limitation, future research could focus on developing sophisticated data quality assurance mechanisms that detect and correct anomalies in data collected from IoT devices before processing. This could include the use of machine learning techniques for anomaly detection, data cleansing algorithms to correct errors in real-time, and methodologies for identifying and mitigating sources of data corruption. Implementing these mechanisms could improve the accuracy of data analysis, decision-making processes, and overall system reliability.

References

[1] S. KOCER, O. DUNDAR, and R. BUTUNER, "Programmable smart microcontroller cards," 2021.

[2] M. N. Bhuiyan, M. M. Rahman, M. M. Billah, and D. Saha, "Internet of things (iot): A review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 474–10 498, 2021.

[3] G. Suciu, M. Anwar, A. Ganaside, and A. Scheianu, "Iot time critical applications for environmental early warning," in *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE, 2017, pp. 1–4.

[4] X. Masip-Bruin, E. Marín-Tordera, G. Tashakor, A. Jukan, and G.-J. Ren, "Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 120–128, 2016.

[5] J. Pan and J. McElhannon, "Future edge cloud and edge computing for internet of things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, 2017.

[6] I. Zhou, I. Makhdoom, N. Shariati, M. A. Raza, R. Keshavarz, J. Lipman, M. Abolhasan, and A. Jamalipour, "Internet of things 2.0: Concepts, applications, and future directions," *IEEE Access*, vol. 9, pp. 70 961–71 012, 2021.

[7] L. Belcastro, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Edge-cloud continuum solutions for urban mobility prediction and planning," *IEEE Access*, 2023.

[8] I. Cohen, C. F. Chiasserini, P. Giaccone, and G. Scalosub, "Dynamic service provisioning in the edge-cloud continuum with bounded resources," *IEEE/ACM Transactions on Networking*, 2023.

[9] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.

[10] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool, and W. Dou, "Complementing iot services through software defined networking and edge computing: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1761–1804, 2020.

[11] R. Ranjan, J. Phengsuwan, P. James, S. Barr, and A. van Moorsel, "Urban risk analytics in the cloud," *IT Professional*, vol. 19, no. 2, pp. 4–9, 2017.

[12] J. Dugdale, M. T. Moghaddam, and H. Muccini, "Agent-based simulation for iot facilitated building evacuation," in *2019 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*. IEEE, 2019, pp. 1–8.

[13] A.-T. Shumba, T. Montanaro, I. Sergi, L. Fachechi, M. De Vittorio, and L. Patrono, "Leveraging iot-aware technologies and ai techniques for real-time critical healthcare applications," *Sensors*, vol. 22, no. 19, p. 7675, 2022.

[14] B. V. Philip, T. Alpcan, J. Jin, and M. Palaniswami, "Distributed real-time iot for autonomous vehicles," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1131–1140, 2018.

[15] S. Kitagami, V. T. Thanh, D. H. Bac, Y. Urano, Y. Miyanishi, and N. Shiratori, "Proposal of a distributed cooperative iot system for flood disaster prevention and its field trial evaluation," *International Journal of Internet of Things*, vol. 5, no. 1, pp. 9–16, 2016.

[16] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, and F. D'Andria, "A survey on iot-edge-cloud continuum systems: Status, challenges, use cases, and open issues," *Future Internet*, vol. 15, no. 12, p. 383, 2023.

[17] A. Verma, S. Prakash, V. Srivastava, A. Kumar, and S. C. Mukhopadhyay, "Sensing, controlling, and iot infrastructure in smart building: A review," *IEEE Sensors Journal*, vol. 19, no. 20, pp. 9036–9046, 2019.

[18] I. Sittón-Candanedo, R. S. Alonso, S. Rodríguez-González, J. A. García Coria, and F. De La Prieta, "Edge computing architectures in industry 4.0: A general survey and comparison," in *14th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2019) Seville, Spain, May 13–15, 2019, Proceedings 14*. Springer, 2020, pp. 121–131.

[19] H. Tianfield, "Cloud computing architectures," in *2011 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2011, pp. 1394–1399.

[20] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[21] A. Čolaković and M. Hadžialić, "Internet of things (iot): A review of enabling technologies, challenges, and open research issues," *Computer networks*, vol. 144, pp. 17–39, 2018.

[22] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "Iot gateway: Bridgingwireless sensor networks into internet of things," in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. Ieee, 2010, pp. 347–352.

[23] D. Giusto, A. Iera, G. Morabito, and L. Atzori, *The internet of things: 20th Tyrrhenian workshop on digital communications*. Springer Science & Business Media, 2010.

[24] K. Baras and L. M. Brito, "Introduction to the internet of things," in *Internet of Things*. Chapman and Hall/CRC, 2017, pp. 3–32.

[25] L. Ericsson, "More than 50 billion connected devices," *White Paper*, vol. 14, no. 1, p. 124, 2011.

[26] F. Duarte. (2024) Number of iot devices. [Online]. Available: μhttps://explodingtopics.com/blog/number-of-iot-devices

[27] C. Wang, M. Daneshmand, M. Dohler, X. Mao, R. Q. Hu, and H. Wang, "Guest editorial-special issue on internet of things (iot): Architecture, protocols and services," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3505–3510, 2013.

[28] H. Ning, *Unit and ubiquitous internet of things*. CRC press, 2013.

[29] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[30] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of iot: Applications, challenges, and opportunities with china perspective," *IEEE Internet of Things journal*, vol. 1, no. 4, pp. 349–359, 2014.

[31] X. Li, R. Lu, X. Liang, X. Shen, J. Chen, and X. Lin, "Smart community: an internet of things application," *IEEE Communications magazine*, vol. 49, no. 11, pp. 68–75, 2011.

[32] S. B. Zahir, P. Ehkan, T. Sabapathy, M. Jusoh, M. N. Osman, M. N. Yasin, Y. A. Wahab, N. Hambali, N. Ali, A. Bakhit *et al.*, "Smart iot flood monitoring system," in *journal of physics: conference series*, vol. 1339, no. 1. IOP Publishing, 2019, p. 012043.

[33] J. Kim, "Energy-efficient dynamic packet downloading for medical iot platforms," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1653–1659, 2015.

[34] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[35] A. Z. Alkar and U. Buhur, "An internet based wireless home automation system for multifunctional devices," *IEEE transactions on Consumer Electronics*, vol. 51, no. 4, pp. 1169–1174, 2005.

[36] Z. Bi, L. Da Xu, and C. Wang, "Internet of things for enterprise systems of modern manufacturing," *IEEE Transactions on industrial informatics*, vol. 10, no. 2, pp. 1537–1546, 2014.

[37] S. Yuvaraj and M. Sangeetha, "Smart supply chain management using internet of things (iot) and low power wireless communication systems," in *2016 international conference on wireless communications, signal processing and networking (WiSPNET)*. IEEE, 2016, pp. 555–558.

[38] S. S. Gill, "A manifesto for modern fog and edge computing: Vision, new paradigms, opportunities, and future directions," in *Operationalizing Multi-Cloud Environments: Technologies, Tools and Use Cases.* Springer, 2021, pp. 237–253.

[39] H. Pang and K.-L. Tan, "Authenticating query results in edge computing," in *Proceedings. 20th International Conference on Data Engineering.* IEEE, 2004, pp. 560–571.

[40] P. G. Lopez, A. Montresor, D. H. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. P. Barcellos, P. Felber, E. Riviere *et al.*, "Edge-centric computing: Vision and challenges." *Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, 2015.

[41] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," pp. 37–42, 2015.

[42] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[43] S. Hamdan, M. Ayyash, and S. Almajali, "Edge-computing architectures for internet of things applications: A survey," *Sensors*, vol. 20, no. 22, p. 6441, 2020.

[44] C. M. Fernández, M. D. Rodríguez, and B. R. Muñoz, "An edge computing architecture in the internet of things," in *2018 IEEE 21st international symposium on real-time distributed computing (ISORC).* IEEE, 2018, pp. 99–102.

[45] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran, "The role of edge computing in internet of things," *IEEE communications magazine*, vol. 56, no. 11, pp. 110–115, 2018.

[46] S. U. Khan, "The curious case of distributed systems and continuous computing," *IT Professional*, vol. 18, no. 2, pp. 4–7, 2016.

[47] A. Sunyaev and A. Sunyaev, "Cloud computing," *Internet computing: Principles of distributed systems and emerging internet-based technologies*, pp. 195–236, 2020.

[48] T. Alam, "Cloud computing and its role in the information technology," *IAIC Transactions on Sustainable Digital Innovation (ITSDI)*, vol. 1, no. 2, pp. 108–115, 2020.

[49] M. Collier and R. Shahan, *Microsoft azure essentials-fundamentals of azure.* Microsoft Press, 2015.

[50] E. Bisong and E. Bisong, "An overview of google cloud platform services," *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pp. 7–10, 2019.

[51] A. Wittig and M. Wittig, *Amazon Web Services in Action: An in-depth guide to AWS.* Simon and Schuster, 2023.

[52] T. R. Kelley, "Optimization, an important stage of engineering design," *The Technology Teacher*, vol. 69, no. 5, p. 18, 2010.

[53] V. J. Shute and D. Zapata-Rivera, "Adaptive technologies," *ETS Research Report Series*, vol. 2007, no. 1, pp. i–34, 2007.

[54] A. B. Pittock and R. N. Jones, "Adaptation to what and why?" *Environmental monitoring and assessment*, vol. 61, pp. 9–35, 2000.

[55] S. Tayeb, S. Latifi, and Y. Kim, "A survey on iot communication and computation frameworks: An industrial perspective," in *2017 IEEE 7th annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2017, pp. 1–6.

[56] M. Casado and N. McKeown, "The virtual network system," in *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 2005, pp. 76–80.

[57] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[58] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[59] M. Wyle, "A wide area network information filter," in *Proceedings First International Conference on Artificial Intelligence Applications on Wall Street*. IEEE Computer Society, 1991, pp. 10–11.

[60] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[61] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013, pp. 15–26.

[62] A. Huang, Y. Li, Y. Xiao, X. Ge, S. Sun, and H.-C. Chao, "Distributed resource allocation for network slicing of bandwidth and computational resource," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[63] I. Sattar, M. Shahid, and N. Yasir, "Multi-level queue with priority and time sharing for real time scheduling," *International journal of multidisciplinary sciences and engineering*, vol. 5, no. 8, pp. 16–17, 2014.

[64] A. P. U. Siahaan, "Comparison analysis of cpu scheduling: Fcfs, sjf and round robin," *International Journal of Engineering Development and Research*, vol. 4, no. 3, pp. 124–132, 2016.

[65] L. P. Damuut and P. B. Dung, "Comparative analysis of fcfs, sjn & rr job scheduling algorithms," *AIRCC's International Journal of Computer Science and Information Technology*, pp. 45–51, 2019.

[66] A. Tanenbaum, *Modern operating systems*. Pearson Education, Inc.„ 2009.

[67] H. Fu, M. Sun, B. He, J. Li, and X. Zhu, "A survey of traffic shaping technology in internet of things," *IEEE Access*, vol. 11, pp. 3794–3809, 2022.

[68] F. Habeeb, K. Alwasel, A. Noor, D. N. Jha, D. AlQattan, Y. Li, G. S. Aujla, T. Szydlo, and R. Ranjan, "Dynamic bandwidth slicing for time-critical iot data streams in the edge-cloud continuum," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 8017–8026, 2022.

[69] A. Sari, "A review of anomaly detection systems in cloud networks and survey of cloud security measures in cloud storage applications," *Journal of Information Security*, vol. 6, pp. 142–154, 04 2015.

[70] L. Junhuai, W. Yunwen, W. Huaijun, and X. Jiang, "Fault detection method based on adversarial reinforcement learning," *Frontiers in Computer Science*, vol. 4, p. 1007665, 2023.

[71] A. A. Torres-García, C. A. R. Garcia, L. Villasenor-Pineda, and O. Mendoza-Montoya, *Biosignal Processing and Classification Using Computational Learning and Intelligence: Principles, Algorithms, and Applications*. Academic Press, 2021.

[72] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. P. van Hasselt, S. Singh, and D. Silver, "Discovering reinforcement learning algorithms," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1060–1070, 2020.

[73] C. Szepesvári, *Algorithms for reinforcement learning*. Springer Nature, 2022.

[74] R. Dearden, N. Friedman, and S. Russell, "Bayesian q-learning," *Aaai/iaai*, vol. 1998, pp. 761–768, 1998.

[75] Z.-x. Xu, L. Cao, X.-l. Chen, C.-x. Li, Y.-l. Zhang, and J. Lai, "Deep reinforcement learning with sarsa and q-learning: A hybrid approach," *IEICE TRANSACTIONS on Information and Systems*, vol. 101, no. 9, pp. 2315–2322, 2018.

[76] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the deep q-network," *arXiv preprint arXiv:1711.07478*, 2017.

[77] T. Tiong, I. Saad, K. T. K. Teo, and H. bin Lago, "Deep reinforcement learning with robust deep deterministic policy gradient," in *2020 2nd International Conference on Electrical, Control and Instrumentation Engineering (ICECIE)*. IEEE, 2020, pp. 1–5.

[78] A. Al-Dulaimy, W. Itani, J. Taheri, and M. Shamseddine, "bwslicer: A bandwidth slicing framework for cloud data centers," *Future Generation Computer Systems*, vol. 112, pp. 767–784, 2020.

[79] T. Weichlein, S. Zhang, P. Li, and X. Zhang, "Data flow control for network load balancing in ieee time sensitive networks for automation," *IEEE Access*, vol. 11, pp. 14 044–14 060, 2023.

[80] I. Maity, S. Misra, and C. Mandal, "Dart: Data plane load reduction for traffic flow migration in sdn," *IEEE Transactions on Communications*, vol. 69, no. 3, pp. 1765–1774, 2020.

[81] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 257–266.

[82] Z. Wen, R. Yang, B. Qian, Y. Xuan, L. Lu, Z. Wang, H. Peng, J. Xu, A. Y. Zomaya, and R. Ranjan, "Janus: Latency-aware traffic scheduling for iot data streaming in edge environments," *IEEE Transactions on Services Computing*, 2023.

[83] H. Khan, P. Luoto, S. Samarakoon, M. Bennis, and M. Latva-Aho, "Network slicing for vehicular communication," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e3652, 2021.

[84] S. Zeuch, A. Chaudhary, B. Del Monte, H. Gavriilidis, D. Giouroukis, P. M. Grulich, S. Breß, J. Traub, and V. Markl, "The nebulastream platform: Data and application management for the internet of things," *arXiv preprint arXiv:1910.07867*, 2019.

[85] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 29–42.

[86] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can {JUMP} them!" in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 1–14.

[87] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "phost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–12.

[88] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo, "Leo: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016, pp. 320–333.

[89] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 221–235.

[90] D. O'Keeffe, T. Salonidis, and P. Pietzuch, "Frontier: Resilient edge processing for the internet of things," *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1178–1191, 2018.

[91] Z. Wen, P. Bhatotia, R. Chen, M. Lee *et al.*, "Approxiot: Approximate analytics for edge computing," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 411–421.

[92] A. N. Abbou, T. Taleb, and J. Song, "A software-defined queuing framework for qos provisioning in 5g and beyond mobile systems," *IEEE Network*, vol. 35, no. 2, pp. 168–173, 2021.

[93] K. Swaminathan, V. Ravindran, R. Ponraj, and R. Satheesh, "A smart energy optimization and collision avoidance routing strategy for iot systems in the wsn domain," in *International Conference on Computing in Engineering & Technology*. Springer, 2022, pp. 655–663.

[94] M. Raval, S. Bhardwaj, A. Aravelli, J. Dofe, and H. Gohel, "Smart energy optimization for massive iot using artificial intelligence," *Internet of Things*, vol. 13, p. 100354, 2021.

[95] M. Humayun, M. S. Alsaqer, and N. Jhanjhi, "Energy optimization for smart cities using iot," *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2037255, 2022.

[96] C. Iwendi, P. K. R. Maddikunta, T. R. Gadekallu, K. Lakshmanna, A. K. Bashir, and M. J. Piran, "A metaheuristic optimization approach for energy efficiency in the iot networks," *Software: Practice and Experience*, vol. 51, no. 12, pp. 2558–2571, 2021.

[97] S. Kumar, O. Kaiwartya, M. Rathee, N. Kumar, and J. Lloret, "Toward energy-oriented optimization for green communication in sensor enabled iot environments," *IEEE Systems Journal*, vol. 14, no. 4, pp. 4663–4673, 2020.

[98] V. M. Kuthadi, R. Selvaraj, S. Baskar, P. M. Shakeel, and A. Ranjan, "Optimized energy management model on data distributing framework of wireless sensor network in iot system," *Wireless Personal Communications*, vol. 127, no. 2, pp. 1377–1403, 2022.

[99] S. Shabana Anjum, R. Md Noor, I. Ahmedy, and M. Hossein Anisi, "Energy optimization of sustainable internet of things (iot) systems using an energy harvesting medium access protocol," in *IOP Conference Series: Earth and Environmental Science*, vol. 268, no. 1. IOP Publishing, 2019, p. 012094.

[100] V. Marinakis and H. Doukas, "An advanced iot-based system for intelligent energy management in buildings," *Sensors*, vol. 18, no. 2, p. 610, 2018.

[101] S. S. L. Preeth, R. Dhanalakshmi, R. Kumar, and P. M. Shakeel, "An adaptive fuzzy rule based energy efficient clustering and immune-inspired routing protocol for wsn-assisted iot system," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–13, 2018.

[102] F. G. Brundu, E. Patti, A. Osello, M. Del Giudice, N. Rapetti, A. Krylovskiy, M. Jahn, V. Verda, E. Guelpa, L. Rietto *et al.*, "Iot software infrastructure for energy management and simulation in smart cities," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 832–840, 2016.

[103] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 683–697, 2021.

[104] N. Kaur and S. K. Sood, "An energy-efficient architecture for the internet of things (iot)," *IEEE Systems Journal*, vol. 11, no. 2, pp. 796–805, 2015.

[105] A. H. Sodhro, M. S. Obaidat, S. Pirbhulal, G. H. Sodhro, N. Zahid, and A. Rawat, "A novel energy optimization approach for artificial intelligence-enabled massive internet of things," in *2019 International symposium on performance evaluation of computer and telecommunication systems (SPECTS)*. IEEE, 2019, pp. 1–6.

[106] O. Almurshed, O. Rana, Y. Li, R. Ranjan, D. N. Jha, P. Patel, P. P. Jayaraman, and S. Dustdar, "A fault-tolerant workflow composition and deployment automation iot framework in a multicloud edge environment," *IEEE Internet Computing*, vol. 26, no. 4, pp. 45–52, 2021.

[107] Y. Zhang, X. Wang, J. He, Y. Xu, F. Zhang, and Y. Luo, "A transfer learning-based high impedance fault detection method under a cloud-edge collaboration framework," *IEEE Access*, vol. 8, pp. 165 099–165 110, 2020.

[108] M. Mudassar, Y. Zhai, and L. Lejian, "Adaptive fault-tolerant strategy for latency-aware iot application executing in edge computing environment," *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13 250–13 262, 2022.

[109] J. P. Dias, T. B. Sousa, A. Restivo, and H. S. Ferreira, "A pattern-language for self-healing internet-of-things systems," in *Proceedings of the European Conference on Pattern Languages of Programs 2020*, 2020, pp. 1–17.

[110] K. Olorunnife, K. Lee, and J. Kua, "Automatic failure recovery for container-based iot edge applications," *Electronics*, vol. 10, no. 23, p. 3047, 2021.

[111] H. Yang and Y. Kim, "Design and implementation of fast fault detection in cloud infrastructure for containerized iot services," *Sensors*, vol. 20, no. 16, p. 4592, 2020.

[112] A. Javed, J. Robert, K. Heljanko, and K. Främling, "Iotef: A federated edge-cloud architecture for fault-tolerant iot applications," *Journal of Grid Computing*, vol. 18, pp. 57–80, 2020.

[113] A. R. Shamshiri, M. Ghaznavi-Ghoushchi, and A. R. Kariman, "Ml-based aging monitoring and lifetime prediction of iot devices with cost-effective embedded tags for edge and cloud operability," *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7433–7445, 2021.

[114] J. Seeger, A. Bröring, and G. Carle, "Optimally self-healing iot choreographies," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 3, pp. 1–20, 2020.

[115] A. Dimara, V.-G. Vasilopoulos, A. Papaioannou, S. Angelis, K. Kotis, C.-N. Anagnostopoulos, S. Krinidis, D. Ioannidis, and D. Tzovaras, "Self-healing of semantically interoperable smart and prescriptive edge devices in iot," *Applied Sciences*, vol. 12, no. 22, p. 11650, 2022.

[116] K. Zhang, W. Huang, X. Hou, J. Xu, R. Su, and H. Xu, "A fault diagnosis and visualization method for high-speed train based on edge and cloud collaboration," *Applied Sciences*, vol. 11, no. 3, p. 1251, 2021.

[117] W. Zhang, J. Wang, G. Han, S. Huang, Y. Feng, and L. Shu, "A data set accuracy weighted random forest algorithm for iot fault detection based on edge computing and blockchain," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2354–2363, 2020.

[118] M. Mudassar, Y. Zhai, L. Liao, and J. Shen, "A decentralized latency-aware task allocation and group formation approach with fault tolerance for iot applications," *IEEE Access*, vol. 8, pp. 49 212–49 223, 2020.

[119] A. Samir and H. D. Johansen, "A self-configuration controller to detect, identify, and recover misconfiguration at iot edge devices and containerized cluster system." in *ICISSP*, 2023, pp. 765–773.

[120] M. Ul Mehmood, A. Ulasyar, A. Khattak, K. Imran, H. Sheh Zad, and S. Nisar, "Cloud based iot solution for fault detection and localization in power distribution systems," *Energies*, vol. 13, no. 11, p. 2686, 2020.

[121] F. Medjek, D. Tandjaoui, N. Djedjig, and I. Romdhani, "Fault-tolerant ai-driven intrusion detection system for the internet of things," *International Journal of Critical Infrastructure Protection*, vol. 34, p. 100436, 2021.

[122] P. Liu, Y. Zhang, H. Wu, and T. Fu, "Optimization of edge-plc-based fault diagnosis with random forest in industrial internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9664–9674, 2020.

[123] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future generation computer systems*, vol. 56, pp. 684–700, 2016.

[124] C.-L. Hu, L.-X. Kuo, Y.-H. Chen, T. Tantidham, and P. Mongkolwat, "Qos-prioritised media delivery with adaptive data throughput in iot-based home networks," *International Journal of Web and Grid Services*, vol. 17, no. 1, pp. 60–80, 2021.

[125] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You can teach elephants to dance: Agile vm handoff for edge computing," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–14.

[126] T. Olsson, E. Lagerstam, T. Kärkkäinen, and K. Väänänen-Vainio-Mattila, "Expected user experience of mobile augmented reality services: a user study in the context of shopping centres," *Personal and ubiquitous computing*, vol. 17, no. 2, pp. 287–304, 2013.

[127] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[128] A. P. Plageras, K. E. Psannis, C. Stergiou, H. Wang, and B. B. Gupta, "Efficient iot-based sensor big data collection–processing and analysis in smart buildings," *Future Generation Computer Systems*, vol. 82, pp. 349–357, 2018.

[129] T. Buddhika and S. Pallickara, "Neptune: Real time stream processing for internet of things and sensing environments," in *2016 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2016, pp. 1143–1152.

[130] P. Bonte, R. Tommasini, E. Della Valle, F. De Turck, and F. Ongenae, "Streaming massif: cascading reasoning for efficient processing of iot data streams," *Sensors*, vol. 18, no. 11, p. 3832, 2018.

[131] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 31–36.

[132] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2002, pp. 1–16.

[133] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, "Coda: Congestion detection and avoidance in sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 266–279.

[134] A. A. Rabileh, K. A. A. Bakar, R. Mohamed, and M. Mohamad, "Enhanced buffer management policy and packet prioritization for wireless sensor network," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8, no. 4, pp. 1770–1776, 2018.

[135] M. M. Hasan, S. Kwon, and J.-H. Na, "Adaptive mobility load balancing algorithm for lte small-cell networks," *IEEE transactions on wireless communications*, vol. 17, no. 4, pp. 2205–2217, 2018.

[136] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[137] J. Pisarov and G. Mester, "The impact of 5g technology on life in 21st century," *IPSI BgD Transactions on Advanced Research (TAR)*, vol. 16, no. 2, pp. 11–14, 2020.

[138] K. Alwasel, D. N. Jha, F. Habeeb, U. Demirbaga, O. Rana, T. Baker, S. Dustdar, M. Villari, P. James, E. Solaiman *et al.*, "Iotsim-osmosis: A framework for modeling and simulating iot applications over an edge-cloud continuum," *Journal of Systems Architecture*, vol. 116, p. 101956, 2021.

[139] L. Smith and M. Turner, "Building the urban observatory: Engineering the largest set of publicly available real-time environmental urban data in the uk." in *Geophysical Research Abstracts*, vol. 21, 2019.

[140] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.

[141] F. M. Brazier, J. O. Kephart, H. V. D. Parunak, and M. N. Huhns, "Agents and service-oriented computing for autonomic computing: A research agenda," *IEEE Internet Computing*, vol. 13, no. 3, pp. 82–87, 2009.

[142] L. Carnevale, A. Celesti, A. Galletta, S. Dustdar, and M. Villari, "Osmotic computing as a distributed multi-agent system: The body area network scenario," *Internet of Things*, vol. 5, pp. 130–139, 2019.

[143] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang, and X. Shen, "Deep reinforcement learning for autonomous internet of things: Model, applications and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1722–1760, 2020.

[144] J. Hribar, M. Costa, N. Kaminski, and L. A. DaSilva, "Using correlated information to extend device lifetime," 2019.

[145] J. Schneider, W.-K. Wong, A. Moore, and M. Riedmiller, "Distributed value functions," 1999.

[146] P. Blasco, D. Gunduz, and M. Dohler, "A learning theoretic approach to energy harvesting communication system optimization," *IEEE Transactions on Wireless Communications*, vol. 12, no. 4, pp. 1872–1882, 2013.

[147] M. Chu, H. Li, X. Liao, and S. Cui, "Reinforcement learning-based multiaccess control and battery prediction with energy harvesting in iot systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2009–2020, 2019.

[148] T. Huang, W. Lin, X. Hong, X. Wang, Q. Wu, R. Li, C.-H. Hsu, and A. Y. Zomaya, "Adaptive processor frequency adjustment for mobile-edge computing with intermittent energy supply," *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7446–7462, 2021.

[149] R. Laidi, D. Djenouri, and I. Balasingham, "On predicting sensor readings with sequence modeling and reinforcement learning for energy-efficient iot applications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 8, pp. 5140–5151, 2021.

[150] T. Szydlo, P. Nawrocki, R. Brzoza-Woch, and K. Zielinski, "Power aware mom for telemetry-oriented applications using gprs-enabled embedded devices-levee monitoring use case," in *2014 Federated Conference on Computer Science and Information Systems*. IEEE, 2014, pp. 1059–1064.

[151] B. Balis, R. Brzoza-Woch, M. Bubak, M. Kasztelnik, B. Kwolek, P. Nawrocki, P. Nowakowski, T. Szydlo, and K. Zielinski, "Holistic approach to management of it infrastructure for environmental monitoring and decision support systems with urgent computing capabilities," *Future Generation Computer Systems*, vol. 79, pp. 128–143, 2018.

[152] T. Szydlo, R. Brzoza-Woch, J. Sendorek, M. Windak, and C. Gniady, "Flow-based programming for iot leveraging fog computing," in *2017 IEEE 26th International conference on enabling technologies: infrastructure for collaborative enterprises (WETICE)*. IEEE, 2017, pp. 74–79.

[153] M. Villari, M. Fazio, S. Dustdar, O. Rana, D. N. Jha, and R. Ranjan, "Osmosis: The osmotic computing platform for microelements in the cloud, edge, and internet of things," *Computer*, vol. 52, no. 8, pp. 14–26, 2019.

[154] M. Chu, H. Li, X. Liao, and S. Cui, "Reinforcement learning-based multiaccess control and battery prediction with energy harvesting in iot systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2009–2020, 2018.

[155] T. Szydlo, A. Szabala, N. Kordiumov, K. Siuzdak, L. Wolski, K. Alwasel, F. Habeeb, and R. Ranjan, "Iotsim-osmosis-res: Towards autonomic renewable energy-aware osmotic computing," *Software: Practice and Experience*, vol. 52, no. 7, pp. 1698–1716, 2022.

[156] A. Chatterjee and B. S. Ahmed, "Iot anomaly detection methods and applications: A survey," *Internet of Things*, vol. 19, p. 100568, 2022.

[157] A. Alrajhi, K. Roy, L. Qingge, and J. Kribs, "Detection of road condition defects using multiple sensors and iot technology: A review," *IEEE Open Journal of Intelligent Transportation Systems*, 2023.

[158] Y.-S. Jeong, "Blockchain processing technique based on multiple hash chains for minimizing integrity errors of iot data in cloud environments," *Sensors*, vol. 21, no. 14, p. 4679, 2021.

[159] T. OConnor, W. Enck, and B. Reaves, "Blinded and confused: uncovering systemic flaws in device telemetry for smart-home internet of things," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, 2019, pp. 140–150.

[160] J. Zhong, F. Zhang, S. Yang, and D. Li, "Restoration of interdependent network against cascading overload failure," *Physica A: Statistical Mechanics and its Applications*, vol. 514, pp. 884–891, 2019.

[161] J. Wu, B. Fang, J. Fang, X. Chen, and K. T. Chi, "Sequential topology recovery of complex power systems based on reinforcement learning," *Physica A: Statistical Mechanics and its Applications*, vol. 535, p. 122487, 2019.

[162] J. Ruhl, "Governing cascade failures in complex social-ecological-technological systems: Framing context, strategies, and challenges," *Vand. J. Ent. & Tech. L.*, vol. 22, p. 407, 2019.

[163] L. Zhang, J. Lu, B.-b. Fu, and S.-b. Li, "A cascading failures model of weighted bus transit route network under route failure perspective considering link prediction effect," *Physica A: Statistical Mechanics and its Applications*, vol. 523, pp. 1315–1330, 2019.

[164] L. Xing, "Cascading failures in internet of things: review and perspectives on reliability and resilience," *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 44–64, 2020.

[165] T. Aung, H. Y. Min, and A. H. Maw, "Enhancement of fault tolerance in kafka pipeline architecture," in *Proceedings of the 11th International Conference on Advances in Information Technology*, 2020, pp. 1–8.

[166] M. Norris, B. Celik, P. Venkatesh, S. Zhao, P. McDaniel, A. Sivasubramaniam, and G. Tan, "Iotrepair: Systematically addressing device faults in commodity iot," in *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2020, pp. 142–148.

[167] F. Fei, Z. Tu, D. Xu, and X. Deng, "Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7358–7364.

[168] A. Belhadi, Y. Djenouri, G. Srivastava, and J. C.-W. Lin, "Reinforcement learning multi-agent system for faults diagnosis of mircoservices in industrial settings," *Computer Communications*, vol. 177, pp. 213–219, 2021.

[169] M. Duarte, J. P. Dias, H. S. Ferreira, and A. Restivo, "Evaluation of iot self-healing mechanisms using fault-injection in message brokers," in *Proceedings of the 4th International Workshop on Software Engineering Research and Practice for the IoT*, 2022, pp. 9–16.

[170] Q. Li, Y. Zhu, J. Ding, W. Li, W. Sun, and L. Ding, "Deep reinforcement learning based resource allocation for cloud edge collaboration fault detection in smart grid," *CSEE Journal of Power and Energy Systems*, 2022.

[171] R. Zhu, L. Liu, H. Song, and M. Ma, "Multi-access edge computing enabled internet of things: advances and novel applications," pp. 15 313–15 316, 2020.

[172] S. Aminikhanghahi and D. J. Cook, "A survey of methods for time series change point detection," *Knowledge and information systems*, vol. 51, no. 2, pp. 339–367, 2017.

[173] M. Gupta, R. Wadhvani, and A. Rasool, "Real-time change-point detection: A deep neural network-based adaptive approach for detecting changes in multivariate time series data," *Expert Systems with Applications*, vol. 209, p. 118260, 2022.

[174] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[175] E. Page, "A test for a change in a parameter occurring at an unknown point," *Biometrika*, vol. 42, no. 3/4, pp. 523–527, 1955.

[176] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, p. 107299, 2020.

[177] M. Zameni, A. Sadri, Z. Ghafoori, M. Moshtaghi, F. D. Salim, C. Leckie, and K. Ramamohanarao, "Unsupervised online change point detection in high-dimensional time series," *Knowledge and Information Systems*, vol. 62, pp. 719–750, 2020.

[178] Y. Li, T. Bao, X. Shu, Z. Gao, J. Gong, and K. Zhang, "Data-driven crack behavior anomaly identification method for concrete dams in long-term service using offline and online change point detection," *Journal of Civil Structural Health Monitoring*, vol. 11, pp. 1449–1460, 2021.

[179] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE access*, vol. 7, pp. 133 653–133 667, 2019.

[180] H. Ikeuchi, A. Watanabe, and Y. Takahashi, "Coverage based failure injection toward efficient chaos engineering," in *ICC 2023-IEEE International Conference on Communications*. IEEE, 2023, pp. 4571–4577.