Methodologies for Managing Big Data Analytics Pipelines for Smart City Applications

Nipun Balan Thekkummal

School of Computing

Newcastle University

"This dissertation is submitted for the degree of"

Doctor of Philosophy

Newcastle University May 2023

To my beloved mother, Leela Balan,

whose unwavering love, support, and sacrifices have been the foundation of my life. Your strength and perseverance have always been an inspiration to me, and your belief in me has made all the difference. Thank you for encouraging me to dream big and standing by me every step of my life. This thesis is dedicated to you with all my love and gratitude.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others except as specified in the text and Acknowledgements. This dissertation contains fewer than 45,000 words, including appendices, bibliography, footnotes, tables and equations, and fewer than 40 figures.

Nipun Balan Thekkummal

May 2023

Acknowledgements

Throughout the research and writing of this thesis, I have been fortunate to receive a tremendous amount of support and assistance from many wonderful people. I would first like to express my heartfelt gratitude to my supervisor, Prof. Rajiv Ranjan. His expertise and guidance have been invaluable in shaping my research topic and methodology, and I am truly grateful for his mentorship. A special thank you goes out to my other supervisors, Prof. Philip James and Prof. Aad Van Moorsel. Your cooperation and the opportunities you provided me to conduct my research and further my thesis have been instrumental in my success.

I must also acknowledge my amazing colleagues at Newcastle University, who have been a constant source of collaboration, encouragement, and assistance. You have all played an essential role in my journey, and I am grateful for your willingness to help me every step of the way.

In addition, I would like to express my deepest gratitude to my family for their wise counsel and sympathetic ears. To my loving wife, Tejaswini, your support, patience, and love have carried me through even the most challenging moments. To my brother Anil Kumar and sister Bijila, your belief in me has been a beacon of strength, and I am truly grateful to have you both in my life. And to my dear mother, Leela Balan, your boundless love, guidance, and wisdom have been the foundation of my accomplishments. You always encouraged me to reach for the stars and believe in the endless possibilities of my dreams.

Lastly, I must thank my incredible friends. Your support, understanding, and companionship have been invaluable as we shared our struggles, discoveries, and happy distractions along the way.

Abstract

Smart cities and Early warning systems rely on complex analytics on the data generated by sensor networks, including IoT and social media. In order to extract value from IoT and social media, a massive volume of heterogeneous data needs to be processed, stored and analysed, which demands a combination of tools from stream processing engines, data lakes and analytics tools. As the data sources are highly distributed and significant in the count, the analytics process is also distributed across different layers. The flow of data from the source and through different subsystems of the IoT and social media depends on various aspects like the frequency of observation, sampling, bandwidth availability, type of analytics processing and location of processing. In IoT networks, the analytical processing is distributed across edge and cloud data centres. The network conditions in the IoT network are prone to be highly variant as it relies on wireless networks operating on radio frequencies like cellular and WiFi. It is quite a common scenario in IoT networks to have frequent bandwidth changes while switching networks and various environmental conditions. Hence, data flow management in IoT networks is an essential aspect of managing the quality of service.

This thesis addresses critical challenges in managing the data flow and analytics pipelines for IoT and social media data. The first part of the thesis explains a tool kit to perform data flow experiments using emulation of a three-layered IoT network. The second part processes algorithms to manage data flow based on the workload and bandwidth between the edge and the cloud data centres. The third part of the thesis explains the design of the data ingestion and analytics part of a landslide early warning system. This section discusses methodologies for orchestrating a real-time streaming data analytic pipeline for social media data.

The main contributions of this thesis are i) A set of algorithms to manage the dataflow in IoT networks. ii) A set of tools for emulating IoT networks using edge processing hardware iii) Methodologies and a framework for the orchestration of streaming analytics pipeline for social media data used in an early warning system.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

The unprecedented growth of cyber-physical systems (CPS) created a revolution by creating new services and applications such as environmental monitoring, early warning systems, healthcare and intelligent infrastructure. IoT has been utilised by smart cities[14] as a way to boost the effectiveness and performance of urban infrastructure. Cyber-physical systems and the people on social media manifest a more extensive network of things and people[49]. As defined[33] by Cisco, the Internet of Everything (IoE) is a networked connection of people, processes, data, and things. IoE has increasingly become the most significant data source, providing unprecedented opportunities for organisations, communities and countries. The combination of sensors, things, and people generates large amounts of data, while the processes and analysis of this data harness its significant value. As per the International Data Corporation (IDC) research forecast[99], the amount of IoT data generated by 2025 is expected to reach 73.1 ZB. The efficient use of data for practical reasons like Early Warning Systems (EWS), Smart Cities, and predictive analysis for planning and mitigation is one of the major difficulties in IoE. While the EWSs require near-real-time inference and decision support capabilities, historical IoE data is utilised for planning and predictive modelling that aid in the development of smart infrastructure.

Early warning systems(EWS) play a significant role in managing the risk of natural hazards[66, 55]. The development of early warning systems and rapid response tools designed to aid in adapting to these future challenges are critical [74]. Since the Indian Ocean Tsunami of 26 December 2004, there has been generally an increase in the interest in developing early warning systems [22]. In a survey [133] of EWS capacities and gaps, the UNISDR placed a strong emphasis on the value of early warnings in minimising catastrophe losses. A successful EWS should be people-centred and have the following four essential elements: risk comprehension, technological monitoring, meaningful warning distribution, and reaction capabilities. Such complex systems demand rich data from various data sources, directly or indirectly related to the hazards of interest. People-centred EWS should be use a combination of physical sensors and people-generated data like social media feeds[133, 55]. Sensor values are analysed using statistical processing models to predict and detect events such as floods and landslides. Meanwhile, data from social media like Twitter and Facebook generated by users in the affected area provide situational awareness. In order to derive insights, historical and real-time streaming data analysis is performed. Differences in the type and volume of data and the historical and streaming nature create challenges [113] in orchestrating heterogeneous data processing environments. Early warning systems will have a significant advantage while using a real-time or a near real-time analysis strategy, as a faster response time potentially could save lives. As the Internet of Everything (IoE) continues to grow and evolve, data flow management in IoT presents multiple challenges that need to be addressed to ensure the efficient and effective use of data for applications such as Early Warning Systems (EWS), Smart Cities, and predictive analysis. Some of these challenges include data heterogeneity, volume, velocity, security, privacy, processing and analysis.

Managing data flow from the source, processing, classifying and storing is a multifaceted process involving numerous steps depending on the application context, the complexity

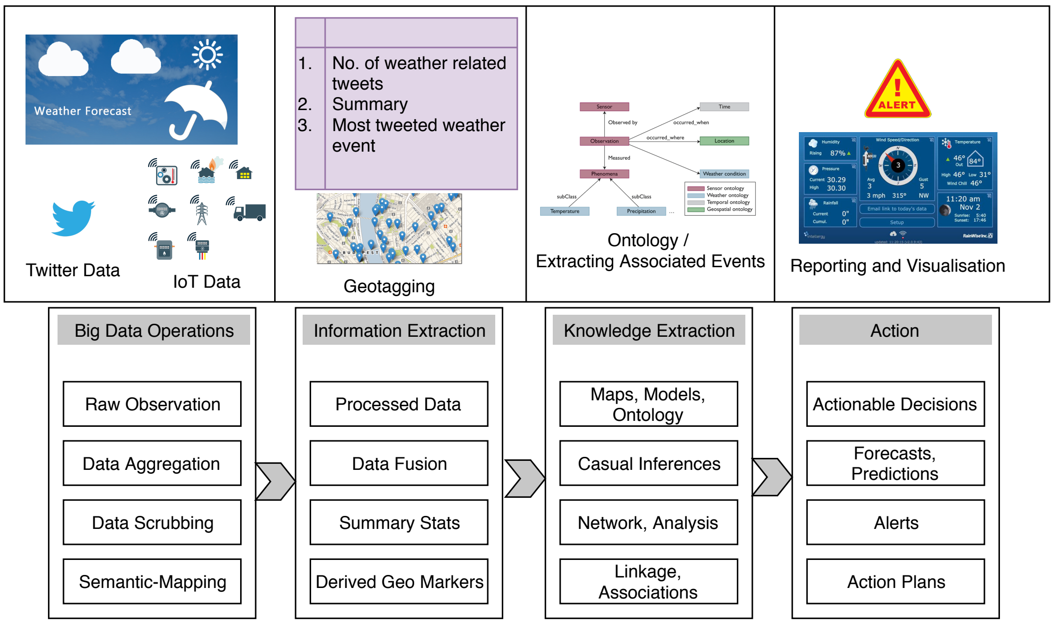


Fig. 1.1 Example of data flow in Early Warning Systems

of analysis and temporal requirements. This thesis addresses several critical problems in managing the data flow and orchestration of real-time data processes for such systems.

For Chapter 3 and Chapter 4, data flow management in a three-layered IoT network is studied. A typical IoT network consists of an IoT layer consisting of sensors and actuators, the edge layer responsible for connecting IoT devices locally and managing data collection, local processing and forwards and the cloud layer, which has the primary analytical and inference workloads. Edge computing enables the processing of data at the edge of the IoT networks, hence improving efficiency by addressing the response time constraints of the cloud layer and bandwidth cost savings. In this research, algorithms have been developed for managing the data flow between the edge and cloud layer. These methods enable local processing and bandwidth management at the edge layer depending on several factors, like the frequency and priority of sensors in the IoT layer. For this, a tool kit has been developed for conducting data flow experiments.

In Chapter 5, data flow in an Early Warning System is studied. The real-time data analytics pipeline for EWS consists of several steps: data collection, cleansing, classification, natural language processing, ingestion and indexing. Investigated methods for orchestrating data processing stages in the analytics pipeline while addressing the main challenges of EWS, like the classification of social media posts.

Evaluation of these methods will be done by developing an analytical pipeline for landslide early warning systems. Pipelines collect, pre-process, classify, store and query social media and public web news site data.

1.1 Research Questions and Challenges

In a broader context, IoT systems' objectives are sensing the physical environment and filtering, sampling and analysing these observed values and extracting meaningful information to perform decision-making. This thesis mainly focuses on the research challenges of data flow management in IoT infrastructure, including sensor data and social media data. These challenges can be classified as data challenges and processing challenges. The data challenges are the volume of the data, the challenge of fusing multiple data observations, the heterogeneity of data by different types of IoT devices, and different data frequencies from various sources. The processing challenges are as follows. i) The complexity of finding a signal from the noise processing data in-stream for near-real-time decision support due to the real-time nature of the IoT data. ii) Orchestrating processing across edge and cloud infrastructure. iii) Workload and Process Distribution in Edge and Cloud

The computing paradigms, such as Edge computing, support data analysis near data sources in IoT use cases. These techniques help manage volume and frequency constraints of data where insights need to be actioned on a timely basis, optimising bandwidth usage. Another challenge is the data fusion complexity from multiple sensors, which is very much domain-specific and context-driven. Sensor fusion is the concept of combining the best information available from each sensor subsystem while ignoring the rest and making inferences and decisions based on it. The challenge of heterogeneous data types also needs to

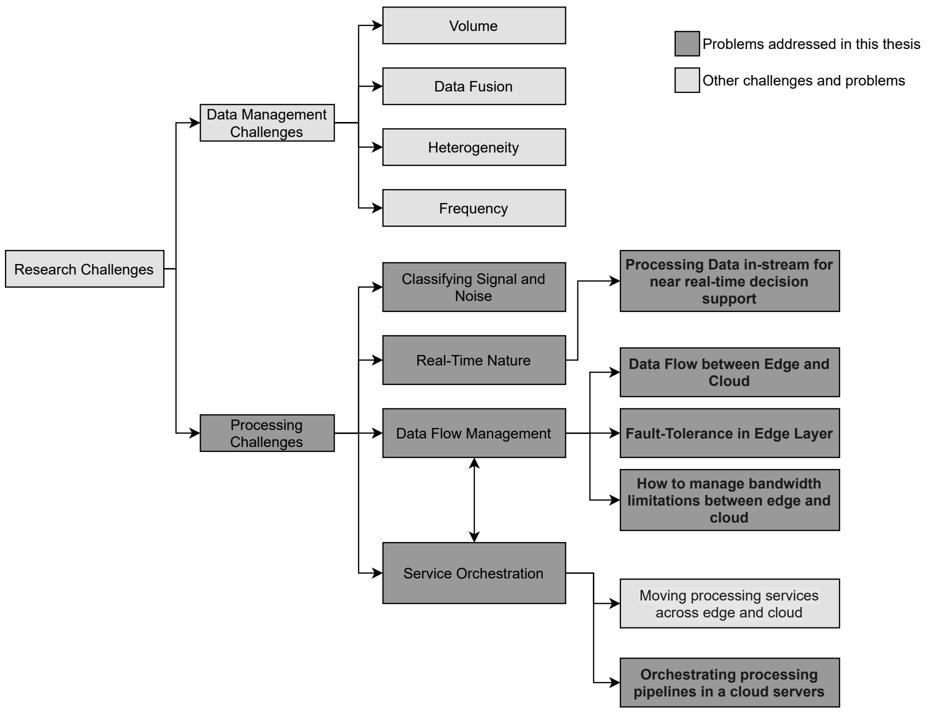

Volume
Data Fusion
Heterogeneity
Frequency
Problems addressed in this thesis
Other challenges and problems
Data Management Challenges
Research Challenges
Classifying Signal and Noise
Real-Time Nature
Data Flow Management
Service Orchestration
Processing Challenges
Processing Data in-stream for near real-time decision support
Data Flow between Edge and Cloud
Fault-Tolerance in Edge Layer
How to manage bandwidth limitations between edge and cloud
Moving processing services across edge and cloud
Orchestrating processing pipelines in a cloud servers

Fig. 1.2 Research Challenges

be addressed based on the context and the domain. This thesis addresses mainly IoT sensor data and social media data.

IoT and social media generate a massive volume of data, and it is a challenge to process and identify a valid signal from the large volume of meaningless information. It is a classical problem of finding a signal from the noise. Depending on the use case, several classification methods are used to classify information based on its value to the purpose. The real-time nature of the IoT data streams is a significant challenge in processing and data flow management. Managing streaming data has a different set of methodologies and QoS parameters. For instance, streaming data are generally timestamped, and the response time for processing the data needs to be optimum. The in-stream analysis is an essential method for processing such data. The scalability of stream processing systems is managed differently and requires more specialised tools than a batch processing system.

Other key processing challenges in IoT networks are data flow management and service orchestration across edge and cloud data centres. These two problems are closely interlinked, where service orchestration methodologies demand data flow management for efficiently moving data according to the service location, availability and bandwidth. This thesis focuses on some key research questions related to data flow management and service orchestration. Data flow management challenges include the data flow control between the edge and the cloud data centres, the decision problem of "process locally" or "forward" to the cloud, and the sampling/filtering strategies. The osmotic computing paradigm addresses critical service orchestration problems like the methodologies to move the workloads and services across the edge and the cloud data centres. Another service orchestration problem is deploying a distributed data processing pipeline for near-real-time processing of IoT and social media data. One of the research questions addressed by this thesis investigates methods to the same with landslide early warning system as a case study.

Following are the research questions addressed in this thesis.

1. What are the methodologies for achieving the maximum data flow rate for each node in the IoT network utilising the available bandwidth while maintaining the stability of the network and the nodes?

2. How to identify data flow bottlenecks in a three-layered IoT network dynamically?

3. What are the methodologies for efficiently coordinating a large number of heterogeneous edge nodes and IoT devices sending data to the cloud simultaneously?

4. How can a distributed analytics pipeline can help in Early Warning Systems like landslide early warning systems and how can the pipelines be efficiently orchestrated?

1.2 Research Contribution

This thesis investigates a comprehensive set of techniques for managing the data flow of IoT data. It also explores methods for processing streaming data and orchestration of the streaming analytics pipeline. The major contributions of this thesis are as follows

1. The first contribution is developing a tool kit for emulating a three-layer IoT network. Conducting real-world experiments in IoT networks requires a workload to run on the same or similar hardware of an edge device. A tool kit has been developed to conduct IoT experiments. It consists of a sensor emulator, Edge Gatekeeper - a middleware software for edge devices, Commander and Adaptive Flow Controller in the cloud, which coordinates the data flow control.

2. The second contribution is a formal model and a set of algorithms for managing data flow for IoT devices. This includes an algorithm to allocate data rates based on the priority of the IoT sensors/devices. This work investigates i) methods to identify backpressure, ii) Algorithms to adaptively adjust data flow rates based on the available bandwidth.

3. The third contribution is the methodology to orchestrate a Big data analytics processing pipeline for social media data processing. This work explains the design of a streaming processing pipeline for an Early Warning System, also investigating the use-case-specific challenges like natural language processing on stream. This work also presents a methodology for identifying bottlenecks in the data streaming pipeline steps.

1.3 Publications

The work presented in this thesis is a result of many collaborations. Following is the list of contributions and publications done during this project.

- Thekkummal, N.B., Jha, D.N., Puthal, D., James, P. and Ranjan, R., 2020. Coordinated data flow control in IoT networks. In Algorithmic Aspects of Cloud Computing: 5th International Symposium, ALGOCLOUD 2019, Munich, Germany, 10 September, 2019, Revised Selected Papers 5 (pp. 25-41). Springer International Publishing.

- Phengsuwan, J., Thekkummal, N.B., Shah, T., James, P., Thakker, D., Sun, R., Pullarkatt, D., Hemalatha, T., Ramesh, M.V. and Ranjan, R., 2019, July. Context-based knowledge discovery and querying for social media data. In 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI) (pp. 307-314). IEEE.

- Phengsuwan, J., TH, N.B. and Ranjan, R., 2019, January. Onto-DIAS: Ontology-based Data Integration and Analytics System for Landslide hazard Early Warning. In Geophysical Research Abstracts (Vol. 21).

- Wen, Z., Phengsuwan, J., Thekkummal, N.B., Sun, R., jamathi-Chidananda, P., Shah, T., James, P. and Ranjan, R., 2020, October. Active Hazard Observation via Human in the Loop Social Media Analytics System. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management (pp. 3469-3472).

- Phengsuwan, J., Shah, T., Thekkummal, N.B., Wen, Z., Sun, R., Pullarkatt, D., Thirugnanam, H., Ramesh, M.V., Morgan, G., James, P. and Ranjan, R., 2021. Use of social media data in disaster management: a survey. Future Internet, 13(2), p.46.

- Contreras, D., Wilkinson, S., Balan, N. and James, P., 2022. Assessing post-disaster recovery using sentiment analysis: The case of L'Aquila, Italy. Earthquake Spectra, 38(1), pp.81-108.

1.4 Thesis Structure

This thesis consists of six chapters. The organisation of the chapters is illustrated in Figure

- **Chapter 1: Introduction**

  This chapter explains the motivation behind the research and the general background of data flow management in IoT networks and social media. It reveals challenges and research questions in managing big data analytics pipelines for IoT networks and early warning systems. The problem, challenges and methodologies are introduced along with the thesis contributions.

- **Chapter 2: Background and Literature Review**

  This chapter presents the background study of IoT and social media data flow management and how EWS use this data effectively. This chapter includes:

  1. Three-layer IoT network architecture details
  2. The role of Osmotic Computing and edge processing in optimising the performance of IoT infrastructure
  3. Real-Time analytics pipelines
  4. The overview of the Landslip project.

- **Chapter 3: Emulation Tool-kit for benchmarking Internet of Things Networks**

  This chapter presents an emulation tool kit for three layers of IoT network emulation. This tool-kit includes tools for emulating sensors, edge processing, edge gateway, edge data flow control, and cloud data ingestion. This toolkit enables to run data flow experiments for three layered IoT networks. This chapter also explains some experiments and their results using this set of tools.

- **Chapter 4: Coordinated Data flow control in IoT Networks**

  This chapter explains models and algorithms to manage data flow in IoT networks which includes i) Formal model of IoT network data flow ii) Algorithm for allocating bandwidth to IoT devices according to priority. iii) Algorithms to adaptively control the data rate of IoT devices as per the network conditions and their evaluation

- **Chapter 5: Managing real-time big data analytics pipeline for Landslide Early Warning System**

  This chapter presents a novel data integration and real-time streaming analysis pipeline for active hazard observation using social media. It shows a prototype system for real-time processing of social media data and their orchestration methods. It addresses some critical issues in the scalability of such pipelines.

- **Chapter 6: Conclusion**

  This chapter concludes the thesis, summarising the contributions and briefly discuss the future direction of the research.

Chapter 2

Background

This chapter describes an overview of the necessary background information and related work to understand better the related technologies, challenges and research questions addressed in this thesis. First, the use of Internet of Things (IoT) data, methodologies and tools are discussed. Furthermore, three-tier IoT network, data flow, and Osmotic Computing concepts are discussed. Finally, real-time streaming analytics techniques in IoT and social media analytics are discussed.

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it". This is a famous statement by the Xerox PARC Chief Scientist Mark Weiser describing ubiquitous computing in his article[138] titled "The Computer for the 21st Century" in Scientific America. He predicted that specialised hardware and software elements connected by wired and wireless networks would be so ubiquitous that no one would notice their presence. The Internet of Things is a convergence of multiple technologies, communication, ubiquitous computing[54] and Big data analytics techniques to make use of the data. Many traditional fields like wireless sensor networks, control systems, embedded systems, and automation helped in the evolution of IoT. IoT is defined as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving

interoperable information and communication technologies" [132] by IoT Global Standard initiative. It describes the network of physical devices consisting of sensors, actuators, software and other technologies for connecting and exchanging data with other devices and systems over the Internet. The devices consist of sophisticated industrial equipment to, household appliances, vehicles and urban infrastructure components. Thus, IoT has become the enabler of a new interaction between people and things and between things by integrating several technologies and communications solutions. It has applications prevalent in logistics, transportation, healthcare, environment, disaster management, smart cities, and urban planning. Things that form the Internet of Things are characterised by low resource capacity in terms of computation and energy [56]. Hence, any IoT solutions should be designed with resource efficiency and scalability in mind.

Application domains of the IoT can be broadly classified[128] into three i) Industry ii) Environment iii) Society. IoT is widely deployed in manufacturing, logistics, banking and finance, governmental bodies and intermediaries in the industry. Companies are investing in redesigning factory workflows, tracking materials and bringing optimisation in every possible way using data analysis. IoT is widely deployed in environmental domains like agriculture and breeding, environment management, recycling and energy management for monitoring, protecting and developing natural resources. Smart farming is already helping many farmers optimise yields, profitability and protection of the environment. IoT is used in activities related to societies, cities, and people's development and inclusion in the Society domain, including smart infrastructure, smart cities, e-participation and inclusion. For instance, IoT-enabled devices like sensors, actuators, and advanced data analytics are helping us reduce air pollution in some of the world's biggest cities and improve agriculture and food supply.

The following sections discuss more on the technology, protocols and devices involved in modern IoT.

2.1 Related Technologies and Use Cases

The following section explains a brief overview and illustration of the Internet of Things communication technologies and protocols. Different protocols are designed to work in different layers.

2.1.1 Wireless Communication Technologies in IoT

Wireless communication technology plays a vital role in the Internet of Things echo system. It enables connectivity between smart devices wirelessly. Wireless technology used in IoT systems has specific characteristics in terms of its capabilities and the requirement for it to work in a resource-constrained environment like IoT. For instance, most of the wireless technologies used in IoT have strict power usage requirements as IoT devices are mostly battery-operated and are deployed away from regular power supply.

Wi-Fi is a popular choice for local networks due to its high data rates and broad coverage, making it ideal for data-intensive applications like video surveillance. Zigbee and Z-Wave are mainly utilised in home automation due to their low power requirements and mesh networking capabilities, enabling devices to communicate through other devices and extending the network's range. Bluetooth, and Bluetooth Low Energy (BLE), is frequently employed for short-range, low-power applications, such as wearables and health monitoring devices. On the other hand, Long Range (LoRa) and SigFox are low-power, wide-area network (LPWAN) technologies designed for long-range communications in applications such as smart cities or agriculture, where devices are dispersed over a large geographical area. Cellular networks (3G, 4G, and 5G) provide extensive coverage and high data rates, making them appropriate for applications requiring real-time communication and high mobility. Table 2.1 lists the important characteristics of IoT's leading wireless communication technologies.

Table 2.1 IoT Communication Technologies

	Frequency Band	Range	Data Rate	Power	IoT Use Cases
NFC	13.56 Mhz	4 cm	106 - 424 kbit/s	Very Low	Cashless Payments, Asset Tracking, Time & Attendance, Transport and Logistics
BLE	2.4 Ghz	100+ m	125 kbps - 2 Mbps	Very Low	Health care, Smart Wearables, Advertisement, Retail, Automotive, Indoor Location and Navigation
ZigBee	2.4 Ghz band	10-100m	20- 250 kbps	Low	Home & Building Automation, Wireless sensor Networks, Industrial Control Systems, Medical Data Collection, Fire and Safety Systems
Z-Wave	Sub GHz bands	15-150m	9.6 - 100 kbps	Low	Smart Home, Home and building Automation
WiFi	2.4, 5, 6 Ghz	100 m to several kms	Upto 1 Gbps	Medium	Smart Home, Home and Building Automation, Connecting complex IoT devices, CCT
LoRaWAN	Sub Ghz bands	5-20 km	0.3- 50 kbps	Low	Asset Tracking, Sending data from sensors deployed in remote areas, Smart Agriculture/Irrigation.
SigFox	868, 902 Mhz	10-40 km	100 bits/sec	Low	Sending data from remote IoT devices and sensor networks
Cellular	900 - 3500 Mhz	Upto 40km	Upto 1 Gbps	High	Connecting Remote IoT devices and smart devices, Smart Cities, Early Warning Systems, Smart Environment Sensors

2.1.2 Communication Protocols

1. **Constrained Application Protocol**

   The Constrained Application Protocol (CoAP)[117] is a specialised web protocol for use with resource-constrained nodes, e.g., low-power, lossy networks. The CoAP is a web transfer protocol based on REST and HTTP functionality. REST is an easier way for clients and servers to exchange data over HTTP. CoAP, an application layer protocol for IoT applications, was developed by the IETF Constrained RESTful Environments (CoRE) working group.

   REST enables servers and clients to expose and consume web services similar to the Simple Object Access Protocol (SOAP) using Uniform Resource Identifiers (URIs) as nouns and HTTP get, post, put, and delete methods as verbs. REST does not rely on XML for message exchanges. CoAP is bound by default to UDP instead of TCP, making it more suitable for Internet of Things applications. In addition, CoAP modifies specific HTTP capabilities to meet IoT requirements, such as low power consumption and operation over lossy and noisy connections. CoAP is based on REST, so conversion between these two protocols in REST-CoAP is straightforward. CoAP aims to make RESTful interactions accessible to low-power, low-computing, and low-communication devices.

2. **Message Queue Telemetry Transport (MQTT)**

   Message Queuing Telemetry Transport (MQTT) is a messaging protocol was introduced in 1999 by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom (now Eurotech) and standardised by OASIS[32, 124] in 2013. Its objective is to connect embedded devices and networks to applications and middleware. The connection operation employs a routing mechanism (one-to-one, one-to-many, and many-to-many). It enables MQTT as the optimal connection protocol for the Internet of Things and machine-to-machine communication.

   MQTT uses the pub/sub pattern and offers transition flexibility with a simple implementation. It is suitable for devices with limited resources and lossy network conditions with low bandwidth. MQTT runs on top of TCP protocol and has 2 QoS levels. There are five major versions of MQTT have five main versions

   (a) MQTT v3.1[79]: Earlier version of the protocol, which is widely in use and well-supported

   (b) MQTT v3.11[123]: This version is an OASIS standard which is built on top of MQTT v3.1. It is one of the last decade's most widely used IoT protocols.

(c) MQTT v5.0[124] The latest version of MQTT as of 2023 is an OASIS standard. It includes a few new features, including shared subscription and improved error reporting mechanisms.

(d) MQTT-SN[125] is an adaptation of MQTT designed to make it more suitable for sensor networks. It is more suitable for scale and power-constrained wireless sensor networks.

3. **Advanced Message Queuing Protocol (AMQP)**

   The Advanced Message Queuing Protocol AMQP is an open standard[122] for transmitting business messages between applications or organisations. It connects systems, provides business processes with the required information, and reliably transmitsÂăthe instructions necessary to accomplish their objectives.

   The main features of AMQP are queuing, routing (point-to-point, pub/sub and fanouts) and message orientation. AMQP is a feature-rich protocol with a higher overhead than MQTT and might not work efficiently in resource-constrained devices. It supports "at most once", "at least once", and "exactly once" message delivery like MQTT. However, these features come at the expense of increased resource consumption and complexity, making AMQP more suitable for enterprise-level applications requiring high reliability and security levels.

Table 2.2 lists and compares more protocols used in IoT networks. Figure 2.1 maps the protocols and the layers in which they operate.

2.1.3 IoT Application Domains

IoT technologies have vast applications that span multiple domains and industries. This section lists the important IoT application domains. Table 2.3 briefly explains the IoT application domains.

Table 2.2 IoT Messaging Protocols

Criteria	HTTP/ RESTFul	MQTT	AMQP	CoAP	WebSockets	DDS	XMPP
Communication Model	Req/Resp	Pub/Sub	Req/Resp, Pub/Sub	Req/Resp, Pub/Sub	Req/Resp, Pub/Sub	Req/Resp, Pub/Sub	Req/Resp, Pub/Sub
Header Size	Undefined	2 Bytes	8 Bytes	4 Bytes	Undefined	Undefined	Undefined
Payload Size	Undefined/ Depends on web server	Upto 250 MB	Undefined	Less than 1 IP Datagram	Undefined	Undefined	Undefined
Transport Protocol	TCP	TCP. UDP	TCP	UDP	TCP	TCP, UDP	TCP
QoS / Delivery Guarantees	TCP based Flow and Congestion Control	At most once, At least once, Exactly once	At most once (Settle Format), At least once (Unsettle Format)	At most once (Confirmable Message), At least once (Non Confirmable Message)	TCP based Flow and Congestion Control	Extensive	None
Encoding Format	Text serialised to Binary	Binary	Binary	Binary	Text serialised to Binary	Binary	XML serialised to Binary
Scope in IoT	Device-Cloud, Cloud-Cloud	Device-Cloud, Cloud-Cloud	Device-Cloud, Cloud-Cloud	Device-Device	Cloud-Device, Cloud-Cloud	Device-Device, Device-Cloud , Cloud-Cloud	Device-Cloud, Cloud-Cloud
Security	HTTPS	TLS	TLS + SASL	DTLS	HTTPS	TLS, DTLS, DDS Security	TLS + SASL

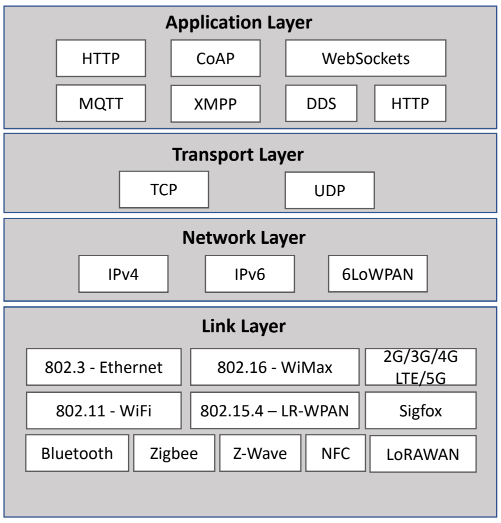


Fig. 2.1 IoT Communication Protocols in Different Layers

2.1.4 Smart Cities

The process of urbanisation has resulted in the emergence of numerous threats, issues, and problems. Only by using "Smartness" can concerned administrations find optimal solutions for the city’s challenges. Moreover, smart cities are more eco-friendly, livable, secure, green,

IoT Application Domains	**Description**
Building and Living	Enhances comfort, energy efficiency, and safety via automation and real-time monitoring systems.
Healthcare	Allows for remote patient monitoring, care for the elderly, and fitness tracking, and provides essential data for diagnosis and preventative care.
Environment	Contributes to environmental monitoring and conservation efforts.
Early Warning and Disaster Management	Enables real-time disaster alerts and post-disaster rescue efforts
Energy	Boosts energy efficiency, facilitates intelligent grids, and integrates renewable energy sources.
Transportation and Logistics	Enhances fleet management, asset tracking, and traffic management while enabling intelligent parking solutions.
Manufacturing	The foundation of Industry 4.0, enabling automation, real-time monitoring, predictive maintenance, and improved supply chain management.
Retail	Improves inventory management and enables automated checkout systems.
Agriculture	Allows for precision farming, automated irrigation, crop health monitoring, and livestock monitoring.

Table 2.3 Application Domains

or connected. Indeed, all of these goals can be thought of as constituting the definition of Smart City. According to Nam and Pardo (2011) "A city can be defined as "smart" when investments in human and social capital and modern transport and communication infrastructure fuel sustainable economic growth and a high quality of life, with a wise management of natural resources, through participatory governance." Fig 2.2 shows the layered architecture of a generic smart city

Challenges of Smart Cities

The idea of smart cities is widely accepted, and many have begun to put it into practice, but there are still problems that must be fixed before the concept can advance any further. There

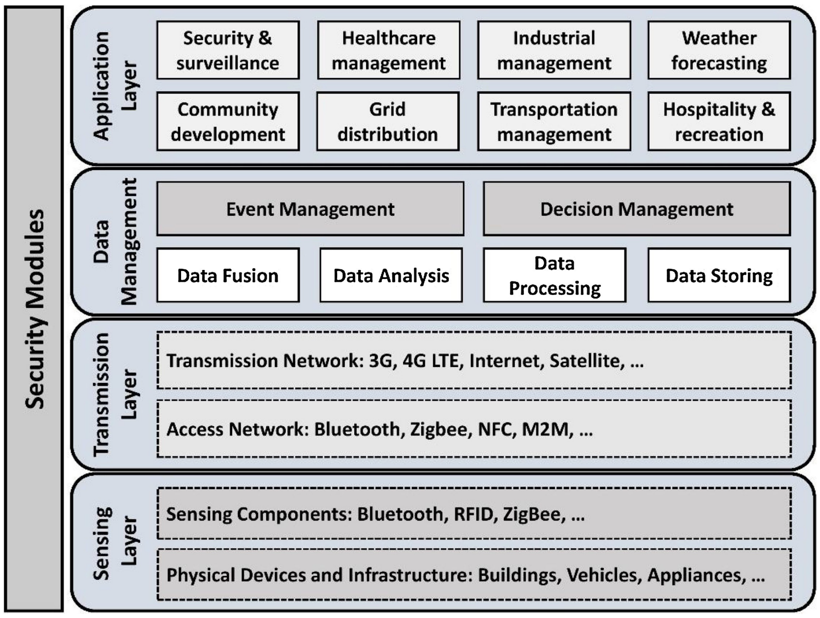


Fig. 2.2 Layered Architecture of a generic smart city[119]

are many obstacles that must be overcome before IoT can become widely adopted, including high design and operation costs, device heterogeneity, massive data collection and analysis, information security, and sustainability.

- Cost: One of the biggest obstacles to the practical implementation of smart cities is the design and maintenance cost. There are two types of expenses: planning and running costs. The initial investment needed to build a smart city is the design cost. Therefore, the lower the design cost, the more likely it will be used in the real world.
- Heterogeneity: The ability to integrate all these disparate components at the application layer is crucial to the practical implementation of the smart city concept. However, integration and inter-operation at the application layer are hampered by platform incompatibilities resulting from heterogeneity.
- Data Privacy: Data privacy is essential in a smart city's infrastructure. Citizens of a smart city can access the city's essential services through their computers, smart-

phones, and other connected devices. In light of this, addressing privacy concerns like eavesdropping is crucial.

- Data Volume: For a smart city to run smoothly and without interruption, it must be able to transfer, store, recall, and analyse large amounts of data. As a result, smart cities need to explore novel areas and promising methods for handling the generation and analysis of Big Data.
- Failure Management: Failure management is also an important consideration for smart city construction projects. System failures, like infrastructure breakdown and network unavailability, can occur after natural disasters like floods, earthquakes, and tornadoes.

Smart City Initiatives

- OASC[9]: Together, the world's smart cities are working on a project called Open and Agile Smart Cities (OASC). The goal of the Open & Agile Smart Cities initiative is to facilitate the development of a community-driven, open smart city market. To remain competitive, cities require interoperability and standards that allow for easy comparison to benchmark performance, sharing best practices, and eliminating vendor lock-ins.
- OrganiCity[91]: OrganiCity is a new European Union project that prioritises citizens in urban planning. Three of the world's most advanced smart cities and 15 experts from a wide range of fields will collaborate on this project.
- UrbanIxD[121] : The goal of UrbanIxD, a two-year European project (FP7-FET), is to establish a network of researchers interested in studying human activity, experience, and behaviour in data-rich urban settings.

2.2 IoT Data Analytics

The IoT constantly produces massive amounts of data from a wide range of devices. Data analytics processes, which can be roughly divided into historical and real-time categories, are used to extract useful information and insights from these datasets. Figure 2.3 shows the IoT data analytics methods and use cases.

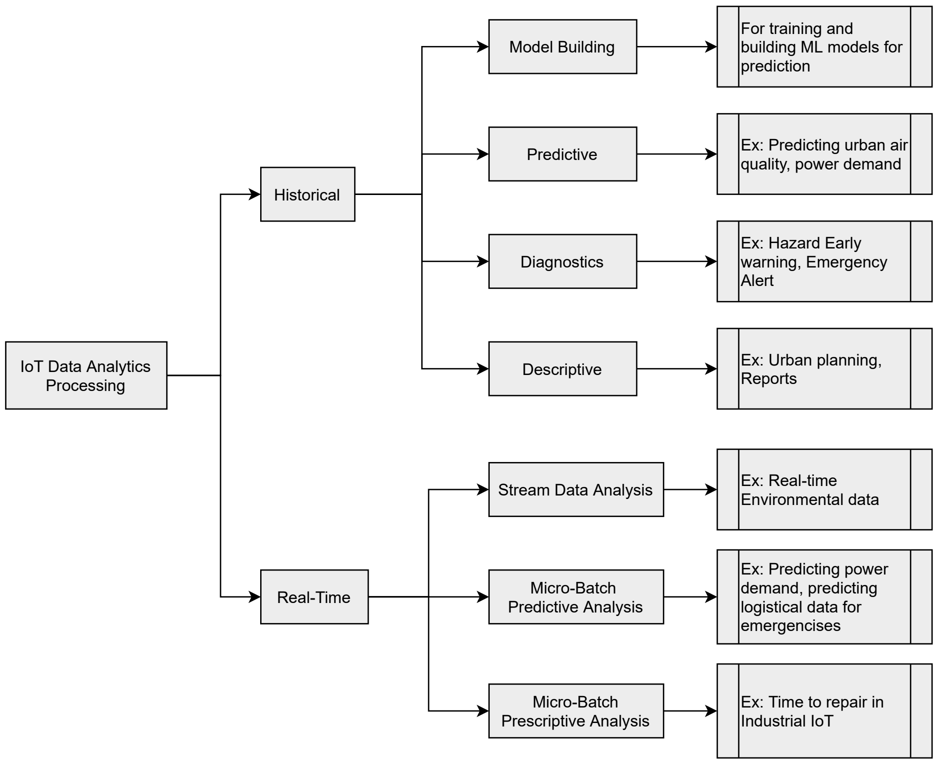

IoT Data Analytics Processing
Historical
Real-Time
Model Building
Predictive
Diagnostics
Descriptive
Stream Data Analysis
Micro-Batch Predictive Analysis
Micro-Batch Prescriptive Analysis
For training and building ML models for prediction
Ex: Predicting urban air quality, power demand
Ex: Hazard Early warning, Emergency Alert
Ex: Urban planning, Reports
Ex: Real-time Environmental data
Ex: Predicting power demand, predicting logistical data for emergencises
Ex: Time to repair in Industrial IoT

Fig. 2.3 IoT Data Analytics

2.2.1 IoT Data Flow Management

Data flow management of IoT systems requires systems to collect, process and store information from the source to the destination. The flow of data is affected by various factors like the data generation rate at the sensors, and the network variabilities due to factors like switching of network and load at the edge data center.

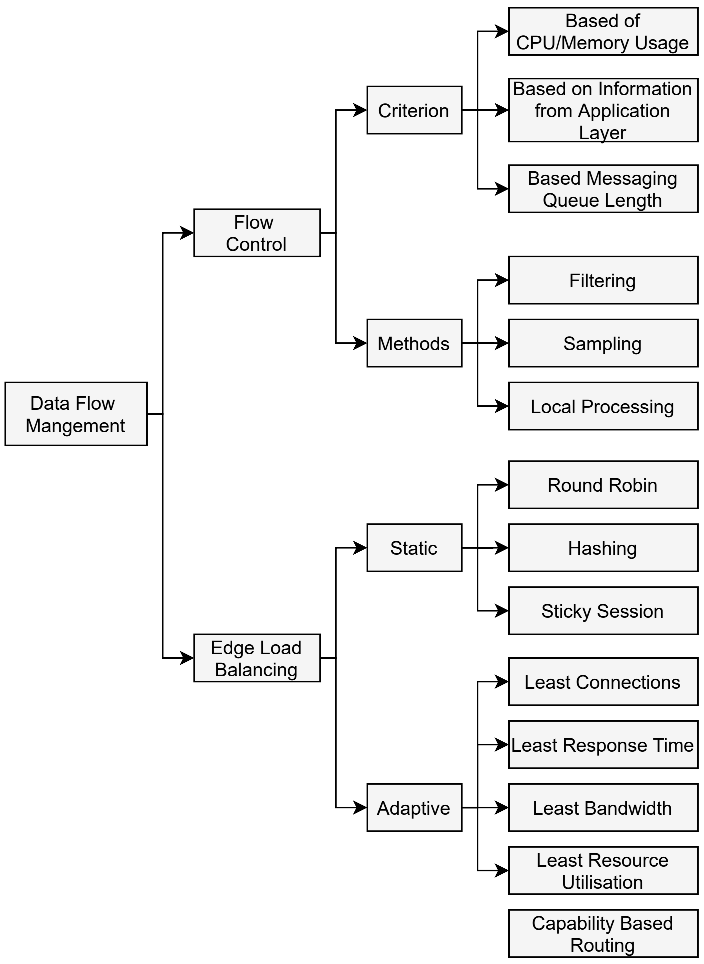


Fig. 2.4 IoT Data Flow Management Taxonomy

2.2.2 IoT Data Flow Challenges

The data flow management challenges in IoT networks are well studied, and both industry and academia have proposed multiple solutions. Lukić et al. (2018) study various methods to establish a data path between sensor nodes and web-or cloud-based IoT applications. In this work, the authors studied the details of data flow challenges in the IoT network while using long-range, low-power networks such as LoRAWAN and NB-IoT. In their paper Szydlo et al. (2017) used the concept of data flow transformation to run parts of the computation closer to the origin of data on edge devices with constrained resources.

Edge computing paradigms address many IoT data flow challenges like latency, bandwidth and network congestion, scalability, resilience and, reliability etc.

Edge Computing Paradigms

Computing methodologies that bring the computation and data storage closer to the source of data, which reduces latency and save bandwidth while improving privacy and security.

1. **Fog Computing**: Fog computing[37], which was developed by Cisco, extends cloud computing to the network's edge. It distributes storage and processing to the most logical and efficient locations between the data source and the cloud. It is capable of scalable management of a large number of nodes.

2. **Cloudlet**: A Cloudlet[17] is a small-scale cloud data centre located at the Internet's edge. It is the middle level of a three-level hierarchy: mobile device cloudlet cloud. Cloudlets are an architectural element that extends the cloud computing infrastructure of the present day.

3. **Osmotic Computing**: Osmotic computing[137] is a new paradigm for deploying and executing microservices across edge and cloud computing infrastructures. The primary

objectives of osmotic computing are to support the efficient execution of Internet of Things (IoT) services and applications within an interoperable, multi-cloud ecosystem.

2.2.3 IoT Data Streams

Stream processing, which started from a single machine system such as Aurora [5] and TelegraphCQ [40], has been studied for decades. With the increasing amount of input data, stream processing has been moved to a distributed processing paradigm, for example, Spark Streaming [140], Storm [131], Flink [38], Google Dataflow [12]. Furthermore, stream processing in the IoT environment requires systems that can utilise the computing resources from both edge nodes and the cloud to achieve low latency and high throughput [139]. In this paper, we re-use the existing sampling technology and priority queue to ensure the QoS when the computing resources are limited. IoT data streams have some desirable properties for their applications:

High throughput. In *edge+cloud* solutions, an edge node may connect with a large number of sensors; similarly, a large number of edge nodes are continuously sending data to the cloud. Therefore, high throughput processing is the key to keeping up with the incoming streams. For example, more than 15,000 sensors (attached to around 1,200 sensor nodes) were deployed in the Santander project for smart city research [43], and Boeing 787 creates half a terabyte of data per flight reported by Virgin Atlantic [Boeing].

Low latency (data freshness). The latency or data freshness is defined as the elapses between a sensor sensing the value from an IoT device and the data arriving at the cloud. Data latency is a crucial performance parameter in the sensor network, as delayed or stale data could result in wrong analysis/decision-making. Consider a real-time traffic monitoring application which collects the traffic data across the city. If the latency is too high because of limited network bandwidth or computing resources of edge nodes, this application may not be able to make a correct decision or may provide inaccurate information for users. Consequently, low latency is very important for this distributed IoT data stream processing system. The system designer should use techniques like sampling [139] and edge processing to ensure low latency in low bandwidth/high latency IoT networks.

Adaptability. Computation resources are geo-distributed in *edge+cloud* architecture, and the underlying hardware are heterogeneous. As a result, the system needs to be adapted to the dynamic IoT environment. For example, cloud-based distributed stream process systems have frequently experienced crash-stop node failures [136, 11], and IoT environments have much more uncertainty than cloud environments [103].

2.2.4 Computational Models for IoT Data Streams

For IoT data, several streaming computation models are preferred depending on the use cases. This section introduces a few computation models in use.

Bulk-synchronous parallel (BSP). This model has been used in many stream processing systems such as Spark Streaming [140], Google Dataflow [12] with FlumeJava. In this model, the computation is a directed acyclic graph (DAG) of operators (e.g. sum, group by, join), the DAG is partitioned into the available computing resources. Moreover, the *mini-batch*, which reserves the streaming data for **T** seconds, is processed over the entire DAG.

Continuous Operator Streaming (COS). In this model, there is no blocking *barrier* like *mini-batch* to sync all operators in a defined time period. These operators are long-running operators, and the messages can be transferred directly among them. The representative systems are Naiad [100], Stream Scope [78] and Flink [38].

Approximate computing. Sampling techniques have been applied in distributed big data analytics to obtain a reasonable output efficiently [7, 57, 69]. This approach is based on the observation that, for many applications, only an approximate value of the sensor reading is required for its desired functionality. Chippa et al. (2013) reported that by reducing 5% accuracy of the k-means clustering algorithm, 50 times energy can be saved. As a result, we leverage a similar strategy by using a sampling algorithm to overcome the trade-off between the limited computing resources and processing the large volumes of IoT data in real-time.

Next, these techniques are widely applied in stream processing [112, 23], which demonstrated well that the proposed systems could balance the quality of output and computation efficiency. Unfortunately, these systems only work on cloud-based environments and cannot utilise the computing resources from edge nodes. Wen et al. (2018) share the same architecture in their work and use the simple technique to overcome the issue of how to sample IoT data in *truly* distributed environments while ensuring low latency and high throughput.

Simple random sampling (SRS). SRS is a naive approach for sampling, in which *n* samples are drawn from a population in such a way that the chance of sampling every set of *n* individuals is equal. Thus, SRS is an unbiased surveying technique which means the selected subset can represent the population as a whole.

Data flow management of IoT systems requires systems to collect, process and store information from the source to the destination. The flow of data is affected by various factors like the data generation rate at the sensors and the network variabilities due to factors like switching of network and load at the edge data center.

2.2.5 Big Data Orchestration

Big Data flow orchestration systems refers to the programming technology to manage the interconnections and interactions among big data workloads on public and private cloud infrastructure. It interconnects tasks into a cohesive workflow to accomplish a goal, with permissions oversight and policy enforcement. Workflow orchestration system *provision, deploy or start servers, tasks and services; acquire and assign storage capacity; manage networking; create VMs; and gain access to specific software on cloud services*. This is accomplished through three main, closely related attributes of cloud orchestration: service, workload and resource orchestration. The main aspects of a workflow management system are workflow model/specification, workflow initiation, scheduling/workflow mapping, and resource management. Several big data processing partially address each of these challenges. Big data processing frameworks use resource managers like Apache YARN [135, 108] and Apache Mesos [53]. Apache YARN provides resource management and a central platform to deliver consistent operations, security and data governance across Hadoop clusters. It has separate daemons for resource management and job scheduling/monitoring. It supports multi-tenancy for data processing and also ensures cluster utilisation by dynamic allocation of cluster resources. While YARN provides resource management for processing frameworks like Apache Hadoop, it does not have a holistic view of the workflow or its resource requirements. It is unaware of the data flow between different workflow stages and assumes HDFS to handle the data locality. Scheduling decision in YARN is taken in a centralised manner. Initially created as a resource manager for Hadoop [118] jobs and later extended to support non-MapReduce applications. Its APIs are complex and provide low level resource management interfaces and it is difficult to build custom YARN based application.

Mesos adopts Non Monolitic/de-centralised scheduling. Mesos does a fine-grained scheduling by providing the frameworks (Hadoop, Spark, Flink etc.) with resource offers and allowing the frameworks to do the scheduling by itself. Each framework won't be having a view of data availability or the flow of data from a workflow perspective. Mesos and YARN are framework-level (Hadoop), and don't have a view of the workflow. Apache Oozie

[Apache] is a workflow scheduler for Hadoop systems. It is a system which runs workflow of inter-dependent jobs. Here, users are permitted to create Directed Acyclic Graphs of workflows, which can be run in parallel and sequentially in Hadoop. Oozie is scalable and can manage the timely execution of thousands of workflows (each consisting of dozens of jobs) in a Hadoop cluster. The scope of Oozie is restricted to the Hadoop environment. Establishes relationship between various data and processing elements on a Hadoop environment and provides feed management services such as feed retention, replications across clusters, archival etc. It accepts entities using DSL (Domain Specific Language), which provides the dependency graph between infrastructure, data and processing logic. Falcon dynamically creates Oozie workflows. Apache NiFi[Apa] was built to automate the data flow between systems. It has a Web-based user interface which helps to design, control, feedback, and monitor. It can be used for enrichment/preprocessing (Parsing, Format Conversion) of data before feeding into analytics platforms like Hadoop or Spark [141]. It supports routing decisions based on the data. NiFi is not used for distributed computing or complex stream processing. It uses a flow-based model using a concept called FlowFile. The preliminary requirement for a workflow engine is to have a language which could describe all the parameters of the workflows. OASIS TOSCA [28, 27] standard enabled the interoperability of cloud application which enhances the portability for cloud applications across different providers.

It provides a vendor-agnostic model to describe an application, its services in cloud, relationship between other services and operational/life-cycle behaviour. Using TOSCA, the structure of an application is represented by a Topology Template, which consists of Node and Relationship Templates. Together, they represent a service as a DAG of deployable components. Qasha et al. (2015) introduces an approach to using TOSCA or describing a scientific workflow. This work used the cloudify DSL to specify scientific workflow which enables automated workflow deployment. The inheritance and object orientedness of TOSCA is good motivation to use as DSL for Big Data Workflow definition. There exist several commercial and open-source configuration management tools like Chef [che], and Puppet [pup]. These tools are capable of managing individual application deployment and configuration. Orchestration engines like Karamel can deploy big data clusters and submit jobs. Orchestration engines in general, serves the purpose of chaining the jobs based on a set of rules.

2.2.6 Message Brokers

There should be a distributed message broker which enables the communication between different processing operators/steps in the pipeline. A broker is centralised; it can receive messages from multiple sources, determine the correct destination and route each message to the correct channel. While the core idea of the broker is centralised, distributed brokers are being widely deployed, which distributed data using data replication and partitioning. Apache Kafka is one of the most widely used as distributed event streaming systems. Messaging brokers serve as the back bone to the system. Every processing steps/operators consume data from the broker and publish its results to the broker in different channels/topic. Message broker should persist data published, and it should be cleared for purging only upon the subscriber demand. This is required for reprocessing, resilience and recovery from an error. Processors and other data sources write into a channel/topic while its subscribers read the information in the order it arrived. This data should be available even after the reading is complete so that it can be re-read in case of any failure or need for reprocessing. This is also useful to perform re-reprocessing of events so that provenance information of a process can be stored. This is especially useful in the case of scientific computing.

Sequential Flow of Data as Events

The data should be entering and exiting the system sequentially, and all operations should be performed in-stream without storing the data beyond the window of operation. This ensures optimal usage of resources of the processing infrastructure. Applications like IoT generate data points based on the reading and the timestamp of observation, which follows this pattern on data flow. Social media feeds also flow sequential data flow based on the time the user interact with the social media. Each message in the system should represent only one event and its data. It should contain only atomic values and should not have repeating groups of data like defined in the 1st normal form of the RDBMS [87]. At the same time, the input and output of the operators can be queued as a buffer for managing varying data processing speeds and network bandwidth.

2.3 Simulator and Emulators for IoT

Simulation and emulation are useful in designing and evaluating IoT systems. Emulation is used for in-depth testing and evaluation of system components and tries to emulate real-world scenarios, while simulation is used for high-level system design and analysis.

Simulation and emulation are crucial research tools for the development of IoT environments. t is often impractical or impossible to test such systems in real-world conditions due to the large number of connected devices and complex interactions characterising IoT networks. The virtual environment simulation and emulation allow for the controlled testing and evaluation of various IoT scenarios like network, configuration and protocols[127]

2.3.1 Simulators

The Cambridge dictionary defines a simulator as: "a bit of equipment that is designed to represent real conditions". A simulator is a system or programming model that can mimic real-world events and identify the outcomes of different hypotheses or assumptions without putting the experimenter in uncertainty, danger or risk[105]. The Internet of Things simulator allows users to simulate networks containing hundreds of user-defined connected devices reliably. Control, configurability, manageability, integration, repetition, and scalability are just a few of the many benefits of using a simulator. The following section discusses a list of 3 leading simulator systems systems.

- **IOTSim**[142]: It is based on the popular CloudSim[36] toolkit, a framework for simulating cloud environments, and adds functionality necessary for the Internet of Things. It's ability to model data processing at the network's edge, closer to where data is generated by IoT devices, can help to alleviate bottlenecks and delay in the transmission of data. Big data processing workflows, such as MapReduce jobs, are frequently used in IoT applications to process large volumes of data generated by IoT devices. IoTSim includes support for simulating such workflows.
- **iFogSim**[60]: iFogSIM is used to model the Internet of Things and Fog infrastructures so that latency, network congestion, energy consumption, and cost can be evaluated as a result of different resource management strategies. In order to simulate the Internet of Things, fog data centres, edge devices, network connections, cloud analysis, and performance metrics assessment, iFogSim is built on top of a cloud simulator specifically designed for IoT. Its features include the ability to compare and contrast different approaches to resource management that have been built around quality of service standards (for things like network congestion, latency, bandwidth, task scheduling, etc.).
- **EdgeCloudSim** [120]: EdgeCloudSim is an open-source simulator used to model and simulate edge computing environments. EdgeCloudSim fills this gap in the domain of edge computing, which requires consideration of both computational and networking

considerations. To assess the many facets of edge computing, EdgeCloudSim offers a mobility model, a network link model, and an edge server model. In addition to its simulation capabilities, EdgeCloudSim also demonstrates improved usability by offering a means by which to retrieve the settings for devices and applications from XML files rather than defining them in code.

2.3.2 Emulators

An emulator is a piece of software or hardware designed to make one system act like another for the purpose of simulation or testing. It is a way to mimic how an Internet of Things (IoT) gadget or system would act in a lab setting. Emulation is the process of running the actual software that would be used in the real IoT system but on a different hardware platform, as opposed to simulation, which uses a mathematical model. This allows for comprehensive testing and evaluation of the system's behaviour, and it can provide a high degree of realism.

Following are examples of some widely used emulators in IoT research

- **MAMMotH**[80]: MAMMotH is a large-scale IoT emulator that can be utilised for the development of experiment scenarios, their subsequent deployment on a test-bed, and the subsequent analysis of the outcomes. It provides link emulation, gateway emulation and node emulation. It uses virtual machines under the hood.

- **COOJA**[18]: In order to simulate the "Contiki OS" and "WSN," the cooja emulation/simulation was developed. It is a java-based simulator and capable of simulating the IoT sensors (Z1 sensors = sense/send/blink), RPL in WSNs (example = UDP-RPL/broadcast), preparation set, system controlled, analysed, and operating system[105]. With COOJA, you can model sensor networks with nodes that are completely unique from one another, both in terms of their on-board software and the simulated hardware. COOJA is flexible because it allows for the simple addition or removal of features and functionality

- **EMULAB** [126]: Emulab is a full-featured network emulation test-bed that allows for complex testing scenarios by virtualising hosts, routers, and networks. Virtual topology allows to emulate resources such as network links. By comparing the virtual topology to the actual hardware, EMULAB determines how best to distribute the available resources.

2.4 Conclusion

This chapter provided comprehensive background information on the key concepts and technologies that form the base of this thesis. Starting with the overall context of the IoT, its architecture, its applications, and their respective domains were all examined. The paper then dove into the details of key wireless communication technologies for the Internet of Things, such as MQTT, AMQP, and CoAP. Several data analytics methods for IoT, both historical and real-time, were also covered in this chapter because of their importance in understanding IoT data. We delved deeper into the difficulties of managing data in the Internet of Things and its data flow.

We investigated several edge computing paradigms, such as Fog Computing and Cloudlet, that are designed to address data flow challenges in IoT. The benefits of these paradigms, as well as their role in mitigating data-related issues, were highlighted.

We discussed the concepts of IoT simulation and emulation, emphasising their significance in IoT research. We discussed various tools useful in this domain, such as IoTSim, EdgeCloudSim, and NetSim. However, it was discovered that there is a gap in the area of IoT emulation, specifically in creating realistic and scalable environments for data flow management research in IoT networks and to conduct data flow experiments.

Chapter 3

Emulation Tookit for benchmarking Internet of Things Networks

3.1 Introduction

According to forecasts[90] by IHS, the number of connected IoT devices will reach 125 billion in 2030. This unprecedented increase in linked networks is due to the spread of IoT devices in the last decade. Due to this rapid increase, the creation of effective and scalable IoT network designs is now necessary to meet the rising needs for data transmission, processing, and security. IoT connects physical devices to the Internet. The data come from various resources, including industrial sensors and control systems, business applications, and open web data and use multiple layers to process and transmit data to the cloud servers. The large volume of big data these devices generate is eventually sent from IoT devices to the cloud for further analysis. This data volume, velocity, veracity, and application contexts bring several grant challenges in IoT application composition, data management, application orchestration, security privacy and compliance [114, 89]. As discussed in Section 1.1, one of the critical challenges in processing IoT data is Data Flow Management, which includes data flow between the IoT, Edge and cloud devices.

Due to bandwidth restrictions, sending data straight to the cloud may only sometimes be efficient or even viable as the volume and speed of data increase. Additionally, the cloud might not be able to handle some time-sensitive applications' need for speedier processing. New computing paradigms that bring processors near the sensors are required to address the issues of high bandwidth, extremely low latency, and privacy concerns. Edge computing[41], Osmotic computing[137, 98], and Fog computing[37, 30] are a few of the well-known concepts created by academia and industry.

IoT sensors send data to the cloud by several means, i.e. (i) Direct: IoT devices can connect directly to the cloud over the Internet using Wi-Fi, cellular, or wired connections. In this approach, devices communicate directly with cloud platforms using IoT-specific protocols (ii) IoT Gateway/ Edge Computing: Edge nodes collect the data from IoT devices and then forward the collected and processed data to the cloud. The former solution needs more flexibility to reprocess the IoT data on the ground due to the limited computing resources of the sensors. In some scenarios, this solution may cause high latency when the network is unstable, wasting network bandwidth to forward the whole data to the cloud, where we only require some aggregated information. The edge and cloud solutions integration has been applied in many state-of-the-art systems [20, 116]. However, these systems consider replacing the analytic jobs close to the data source, reducing latency. ApproxIoT system [139] was designed to utilise edge computing resources such as mobile phones, network gateways and edge data centres at ISPs where the approximate computing is performed by achieving low latency analytics. However, the IoT network's uncertainty affects edge devices' stability and communication to the cloud. For example, the available bandwidth of a cellular network is highly variable due to the changing number of devices connected to a base station. Also, the cellular network is dynamically changing among 2G, 3G and 4G in rural areas with insufficient base stations.

In many cases, high-frequency data is not required when no significant variance in the value or no imminent real-world situation demands a high data rate reading and sensor data analysis. The sensor types, the location and real-world scenarios are the factors which determine the data-sending rate or frequency. For example, a sensor installed for measuring the ambient temperature of a factory floor may send its values twice a second. In contrast, a vibration sensor installed in a compression pump at a critical location sends data at a rate of 100 samples per second. Any significant change in vibration within a short period can cause severe damage if it is not acted upon quickly.

In order to evaluate the real-world data flow in a three-layered IoT network, taking into consideration factors such as data flow, data rates, frequency, and dependability, an IoT emulation toolset is required. It is essential to assess the network's capacity to handle various data speeds and frequencies due to IoT devices' diversity and varied data transmission needs [13]. Researchers and developers may test the network's ability to handle the various data rates and frequencies produced by these devices by simulating the complex data exchanges between heterogeneous devices in the edge, fog, and cloud layers using an IoT emulation toolkit. Additionally, the toolkit offers a way to assess the system's dependability in terms of data freshness, guaranteeing that IoT data is effectively sent and processed with little packet loss and delay. An IoT emulation toolkit is an essential resource for improving system performance, testing novel solutions, and ultimately building a more robust and effective IoT ecosystem that can handle various data types and satisfy the requirements of multiple applications. This thesis investigates a set of techniques for managing the data flow in an IoT network. For this purpose, an IoT Emulation tool kit has been developed. This chapter describes the IoT Emulation toolkit, which consists of a Sensor Emulator, Edge GateKeeper and Adaptive Flow Controller.

The remaining chapter is organised as follows. Section 5.2 sets the background and reviews some related work. Section 3.4 explains the proposed system architecture, including

Sensor emulator, Edge GateKeeper (3.4.2) and Adaptive Flow Controller (AFC) (3.4.3). The implementation details in Section 3.5 and the evaluation of the toolkit with experiments to detect back pressure and data freshness are provided in Sections 4.4

3.2 Background

Smart cities, healthcare, transportation, and manufacturing are just a few of the industries that have seen a substantial influence due to the explosive growth of the Internet of Things (IoT) networks. The creation of cutting-edge tools and techniques is necessary due to IoT networks' complex and diversified nature and the difficulties in design, implementation, and performance evaluation. Setting up real-world IoT test beds may be expensive and time-consuming, which restricts the ability to assess network performance in various scenarios. In light of these difficulties, IoT emulation toolkits [104, 81, 106] have become vital for streamlining the creation, improvement, and assessment of IoT network solutions. Researchers and developers may recreate intricate data flows and communication patterns using emulation toolkits, emulating real-world situations in a controlled setting. In order to guarantee the scalability, dependability, and general performance of IoT networks, benchmarking is crucial. Due to the shortcomings of current performance assessment approaches, assessing network performance under various real-world circumstances can take time and effort. Emulation toolkits provide customised and expandable environments that closely mirror existing IoT networks to overcome these constraints. This makes adopting new IoT technologies and standards easier to adopt, ensuring that networks stay effective and current.

Emulating adaptable and customisable environments enables the development of IoT network designs and communication protocols, enabling networks to adapt to shifting needs and technology[77]. Emulation toolkits will be essential to the success and broad acceptance of IoT networks as they develop further. It enables the effective assessment and optimisation of IoT networks by offering a controlled environment that simulates real-world conditions, thereby aiding their success in various applications. The design and implementation of an emulation toolkit, as well as its potential, is covered in this chapter.

3.2.1 *Edge+cloud* IoT architecture

The increasing number of sensors or IoT devices in smart manufacturing facilities and smart cities are continuously generating and emitting high volumes of data across distributed infrastructures. Further, more powerful devices such as Raspberry Pi, edge gateway, PC or edge cloud collect the emitted data and perform the data aggregation, sampling, filtering, projections or transformation into other formats over these collected data in the edge. This processed data is forwarded to the cloud for more complex analysis. This processing workflow runs over the *Edge+cloud* architecture as shown in Figure 3.1. The *Edge+cloud* architecture helps in reducing latency, increasing privacy and saving network bandwidth [95] compared to the architecture where the sensors or IoT devices are connected directly to the cloud. Fog computing[30] uses *Edge+cloud* architecture to extend the cloud computing paradigm to the edge layer by moving some of the processing workloads to the edge layer.

3.3 Research Questions

In this work, we build a data flow emulation toolkit that can help conduct experiments on three-layered IoT infrastructure. It can test the data forwarding from edge to cloud based on the uncertainty of the IoT network. To this end, we develop a *i) Sensor emulator* which emulates various sensors sending data at different data rates. *ii) Edge Gatekeeper (EGK)* which monitors the status of each edge node while dynamically adapting the data ingestion rate from the IoT sensors to edge nodes, and edge nodes The design of the *EGK* allows any flow algorithms to be implemented and deployed on the edge node like processing and delivering only a subset of the data, sampling and windowed calculations to reduce the latency.

In order to address the following research challenges, the proposed toolkit provides a platform for running data flow experiments:

- What are the methodologies for achieving the maximum data flow rate for each node in the IoT network utilising the available bandwidth while maintaining the network's and the nodes' stability?
- How to dynamically identify data flow bottlenecks in a three-layered IoT network?
- What are the methodologies for efficiently coordinating many heterogeneous edge nodes and IoT devices sending data to the cloud simultaneously?

This work makes the following contribution.

- We propose a system set of tools to emulate different layers on an IoT infrastructure.
- We validated our proposed tool kit's system architecture through experiments using real-world data. Our experimental results support how effectively we can conduct experiments on the real edge and cloud infrastructure, test different data flow control algorithms, and measure conditions like back pressure and rate limiting.

3.4 Architecture

The Emulation tool kit consists of i) A Sensor Emulator ii) Edge GateKeeper, and iii) Adaptive Flow Controller. The system uses MQTT[76] protocol to communicate between the layers. The purpose of the flow controller is to actively control the rate of data being processed and forwarded from the edge device to the cloud. The edge gateway can reconfigure IoT sensors to set the data rate, reducing the amount of data sent by the sensors. This acts as

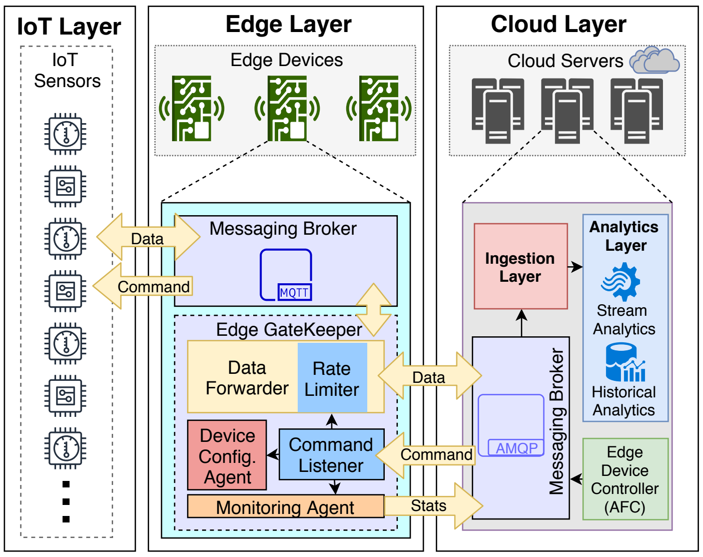


Fig. 3.1 Architecture

a stabilisation mechanism for the varying workload at the edge layer. The sensors' type, frequency, and priority were considered while designing the reference architecture (Figure 3.1). The scenario of a single-hop star network where the sensor IoT devices interact directly with the edge device was evaluated. It consists of 1) the IoT layer, which contains the IoT devices/sensors having the lowest processing capacity; 2) the Edge layer, which consists of devices which act as the entry points to the larger cloud network and is also capable of doing some processing of the sensor data; 3) Cloud layer with the highest processing capacity of all the other layers. Data flows from the IoT to the cloud layer through the edge layer. The data flow control is done using Edge GateKeeper (EGK), deployed in the edge layer and Adaptive Flow Controller (AFC), deployed in the cloud layer. This work proposes the system architecture for dynamically reconfiguring the edge device based on the number of devices connected and the priority of each sensor in the network.

3.4.1 Sensor Emulator

The IoT network's first layer consists of sensors that sense the physical environment and send the observed value to the cloud layer through the edge layer. While the edge layer performs specific actions based on the observed value, the sensor layer makes minimal decisions and actions. For performing experiments on the data flow, it is essential to see data flowing from various kinds of sensors sending data at different frequencies. While we use real hardware like Raspberry Pi for emulating the edge layer, it is very hard to do the same in the sensor layer as there could be a wide range of sensors measuring different physical observations working over various types of networking protocol, data rate and reliability. For instance, a building temperature sensor may operate over a Zigbee protocol, sending data to the edge layer every 1 minute. In contrast, a water pump vibration sensor sends observations ten times every second over Bluetooth. To emulate this situation, we developed a sensor emulator which can emulate 10s to 100s of sensors from a single Raspberry Pi.

Sensor emulator is a critical component of the Emulation tool kit, which enables the researcher to run several sensors from a single hardware. Different types of sensors can be created with different data rates, sending various physical observations. The tool can read files consisting of the sensor readings and send them to the edge layer. It can be configured to read files from the file system and send the observation at different data rates for each sensor. We used real-world observations from the Urban Observatory and Urban Science building[67] data. Sensor emulator uses MQTT[76] messaging protocol for transferring data to the Edge layer where the broker is configured to accept the messages.

3.4.2 Edge GateKeeper (EGK)

An extensive IoT network contains 1000s of sensors connected to 100s of edge devices. The load on the edge device is proportional to the volume and velocity of data. Edge devices can send the whole data as-it-is to the cloud layer or perform sampling/pre-processing before sending it to the cloud. The proposed reference architecture uses a coordinated data flow control to allow the edge devices to balance the data flow by dynamically reconfiguring sensors to send data at different frequencies, allowing control of the volume and velocity of

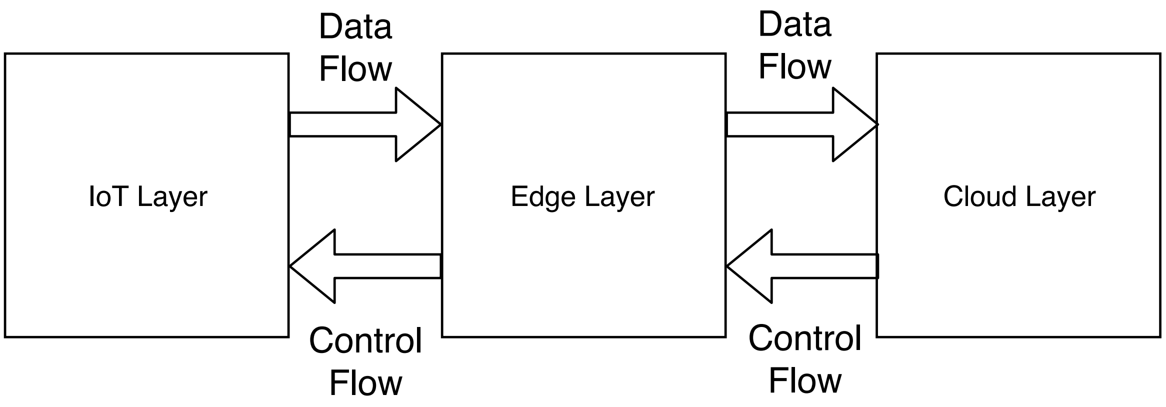


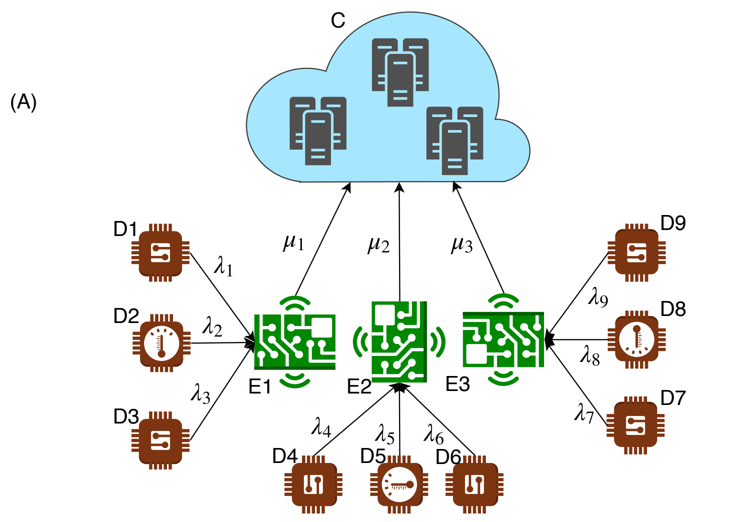
Fig. 3.2 **Data flow and control flow between the layers:** For IoT sensors, data flow happens from IoT layer to cloud while the control flow is from cloud layer to IoT layer

data. This acts as a stabilisation mechanism for a varying workload at the edge layer. The reference architecture consists of IoT sensor devices with a data channel and control channel for communication with the edge device. The edge device is a low-powered computer with essential traffic management and processing capacity.

Edge GateKeeper (EGK) is a lightweight application running on the edge device which acts as the gateway for all communication to and from it. It consists of i) a Lightweight local message broker, ii) Sensor data ingest and Forwarder, iii) a Pre-processor, iv) Rate Limiter, v) a Command listener, and vi) a Device reconfiguration agent. The Sensor data reader reads the data from the message broker to which sensors send their readings in real-time. The pre-processor performs essential pre-processing and filtering operations. Figure 3.4 explains the functionality of EGK.

3.4.3 Adaptive Flow Controller (AFC)

The Adaptive Flow Controller (AFC), hosted in the cloud layer, controls the reconfiguration decisions. EGK and the cloud layer are connected using data and control channels. The data channel carries data forwarded by the EGK from the IoT layer to the cloud ingestion layer. The control channel has commands to control the edge layer and IoT layer. Sensors are registered to EGK, and EGK registers itself and its connected sensors to the ingestion layer in the cloud. The AFC stores the full map of the IoT network as a graph. The graph can be represented as a 2D sparse matrix M (Figure 3.3). Edge devices are represented as rows, and IoT devices are represented as columns. The value at $M[i][j]$ denotes the data rate of IoT device D_j to edge device E_i. Three scenarios can trigger a reconfiguration. i) More IoT devices are added to an edge device. ii) Edge devices' processing capacity becomes a bottleneck for the data flow. iii) There is a demand for higher data rates by some of the sensors. The maximum forward rate is set on the edge device based on the resource capacity of the edge device and network stability. We assume this value is set for each edge device during the initial setup. When additional IoT devices are added, the edge device informs the AFC. The AFC, in turn, recalculates the data rate allocation for each IoT device and sends it back to the edge device. The data rate allocation problem is formally defined in Chapter 4, Section 4.3.



(B)

$$M = \begin{array}{cccccccccc} \text{D1} & \text{D2} & \text{D3} & \text{D4} & \text{D5} & \text{D6} & \text{D7} & \text{D8} & \text{D9} & \\ \lambda_1 & \lambda_2 & \lambda_3 & 0 & 0 & 0 & 0 & 0 & 0 & \text{E1} \\ 0 & 0 & 0 & \lambda_4 & \lambda_5 & \lambda_6 & 0 & 0 & 0 & \text{E2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \lambda_7 & \lambda_8 & \lambda_9 & \text{E3} \end{array}$$

Fig. 3.3 Shows data flow from sensor network to cloud through edge. D1-D9: IoT sensors, E1-E3: Edge Devices, C: Cloud Layer. λ_1-λ_9: Data from from D1-D9 to the connected Edge devices. In matrix M rows represent the edge devices (E1-E3) and columns represent IoT devices (D1-D9)

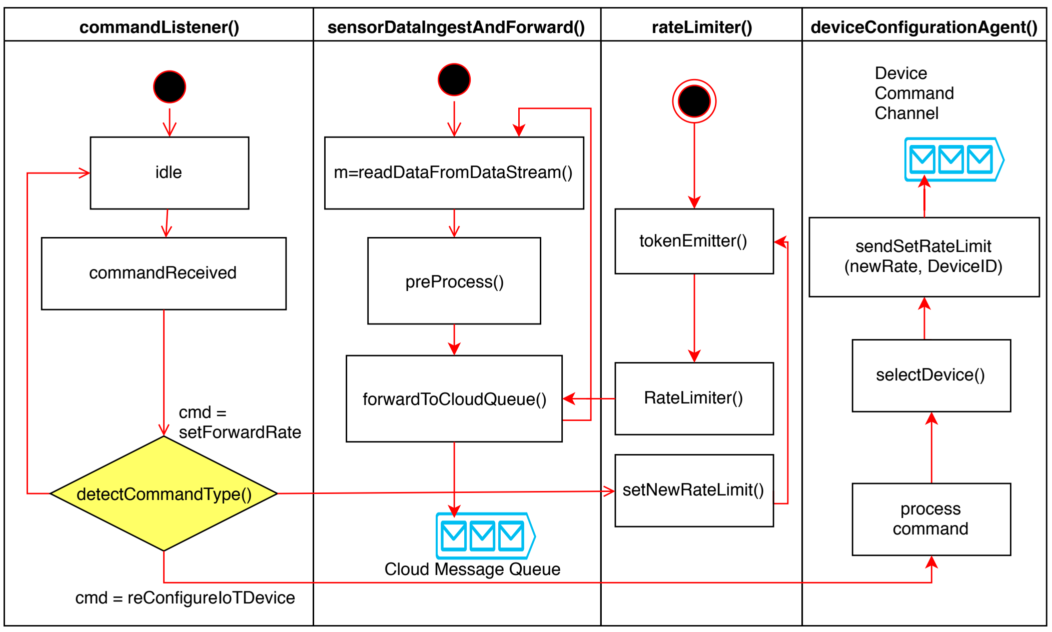


Fig. 3.4 Edge GateKeeper Workflow

3.5 Implementation Details

We used Raspberry Pi 3B+ as edge nodes and for emulating sensors. Raspberry Pi is a credit-card-sized single-board computer (SBC)[134] mainly developed for simulation and teaching purposes. As it has a small form factor, is reliable and has low power requirement, this device is increasingly being used for industrial and IoT applications [96].

We set up the tool kit, consisting of three layers, namely, IoT layer, Edge layer and Cloud layer. Real sensor values from Newcastle Urban Observatory (UO) [67] have been used, an IoT-based city environment monitoring system consisting of about 1000 sensors forwarding around 5000 data points a second. To test this system, the sensor emulator was developed, which sends data points to edge devices at a configured frequency. The sensor emulator runs on three Raspberry Pi devices, each emulating a subset of sensors. Each instance ran a version of the sensor emulator configured with different sensor types, source files, and data rates.

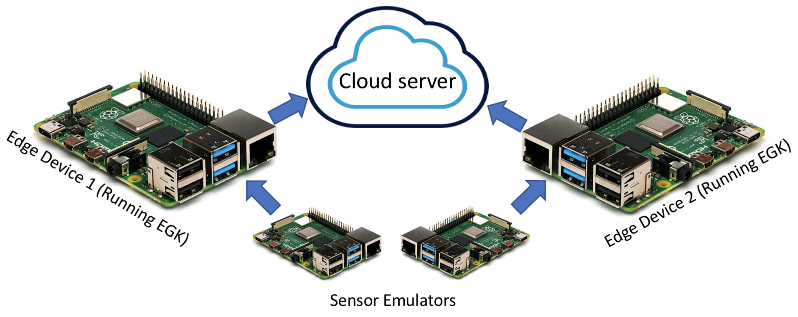


Fig. 3.5 Prototype Implementation

The system is implemented based on the reference architecture (Figure 3.1) proposed in this paper. We used Raspberry Pi 3B+ devices as the edge nodes. Devices are installed with Raspbian OS running in shell-only mode. A lightweight MQTT broker, Mosquitto [76] is installed in each edge device to queue data from the sensors. As mentioned in the architecture section, sensors are partitioned, and each partition is connected to one Raspberry

Pi device through LAN connectivity. An instance of Edge Gatekeeper (EGK) software is configured to run in each edge node. EGK is designed to run with minimal memory and CPU footprint and is developed in GoLang [92]. One topic is created for each sensor for command communication, while a single topic is used for data communication from sensors to the edge. Sensors receive commands from the device configuration agent in EGK by subscribing to the command topic to which it publishes reconfiguration commands. Sensors send the readings as streams of data points in JSON format to the MQTT broker. As MQTT is a pub/sub-based messaging protocol working over TCP/IP, both (emulated) sensors/IoT devices and edge devices communicate through network sockets. We selected a QoS Level 1 [65], which promises "AT LEAST ONCE" delivery assurance to ensure delivery of every message. Elevating to QoS Level 2 has a significant performance impact. All the sensors publish data points to the topic "sensor data". Upon startup, EGK initiates the data forwarder sub-service as a thread which subscribes to the topic "sensor data". EGK receives the messages from all the sensors publishing to its local broker at the configured publish rate. EGK receives data from all the sensors connected to it, and it forwards this to the cloud messaging queue. The data forwarding rate is preset at each edge device, which can be reconfigured by the Adaptive Flow Controller (AFC) in the cloud data centre.

EGK also runs a service which listens to the commands from the (AFC). We have defined reconfiguration commands (Table 3.1) to set the data rate/frequency of both IoT devices and edge devices. A reconfiguration command could be addressed to an IoT device (using sensor_id) or edge device (using edge_id). AFC sends the reconfiguration command to the corresponding edge node. The device reconfiguration agent running as a service in the edge gatekeeper is responsible for reconfiguring the edge and IoT nodes. If it is a command addressed to the edge node, it reconfigures itself with the new data rate. If it is addressed to a sensor, the device configuration agent sends the reconfiguration command to the sensor, which sets the new data rate and continues its normal operation.

Table 3.1 Device Reconfiguration Commands

Command	Description
SET E_FWD_RT $\langle edge\_id \rangle \langle datarate \rangle$	Set data forwarding rate of an edge node
SET S_RT $\langle sensor\_id \rangle \langle datarate \rangle$	Set data rate of a sensor node/IoT device
START E_FWD $\langle edge\_id \rangle$	Instructs an edge node to start forwarding data
STOP E_FWD $\langle edge\_id \rangle$	Instructs an edge node to stop forwarding data
START S_SND $\langle sensor\_id \rangle$	Instructs a sensor node to start sending data points
STOP S_SND $\langle sensor\_id \rangle$	Instructs a sensor node to stop sending data points

AFC is a program running in the cloud environment which has the information of all the sensors and edge nodes. The data rate allocation algorithm (Algorithm 1) is triggered whenever a new sensor is added to the network.

3.6 Experimental Settings

This section describes the experiment settings and configurations used to evaluate the system.

3.6.1 Hardware and Network Configuration

The hardware configuration consists of 2 sensor emulators, each emulating three sensors and two edge devices. All the devices are implemented on Raspberry Pi 3B with 1 GB of RAM. The sensor emulators and edge devices are connected to the network using wired LAN cables through a 1 Gigabit LAN switch. A router is connected to the switch, providing edge layer access to the cloud layer as shown in Figure 3.5. The router is connected to the internet using a 4G network. The 3G and GPRS networks are emulated using the TC [15] Linux tool.

3.6.2 Software Configuration

The edge devices run EdgeGateKeeper, designed to preprocess and forward the data received from the sensor emulators to the cloud layer. The clocks of all devices are synchronised using a standard Network Time Protocol (NTP) server to ensure accurate timestamping of the data records. Sensor emulator listens to the command channel for traffic rate adjustments.

3.6.3 Data Flow and Rate Configuration

Data rates were incrementally set in the experiments on the IoT devices, varying the total data rates from 10 to 450 records per second for each sensor. This configuration results in a total data rate ranging from 100 to 4500 records per second.

In the backpressure detection experiment (Section 3.7.2), each edge device is set with a max forward rate of 5000 records per second. The data rates of sensors are elevated gradually to identify the point at which the edge cannot forward all the data received.

3.6.4 Latency Measurement

Latency is calculated by comparing the epoch time attached by the sensor emulators with the timestamp attached by the edge devices when forwarding the data to the cloud. The latency is tested under three network scenarios: 4G, 3G, and GPRS.

3.6.5 Performance Metrics

The performance of the system is evaluated using the following metrics:

- Forward rate: The rate at which the edge device forwards the received data to the cloud layer.
- CPU usage: The percentage of CPU utilisation by the edge device.

- Memory usage: The percentage of memory utilisation by the edge device.
- Latency: The time difference between the epoch time attached by the sensor emulator and the timestamp attached by the edge device when forwarding the data to the cloud.

These metrics are used to evaluate the performance of the edge device under different conditions and understand the system's limitations in data handling capacity, resource utilisation, and data freshness.

3.7 Evaluation

To validate this system, three experiments were designed to run on the prototype. The primary objective of the experiments is to understand the impact of a different data flow rate for a fixed load on the edge device. The second objective is to understand the point at which the edge device is rendered into an unstable state, where data backpressure starts building on the edge device. The third objective is understanding how adaptive flow control deals with the backpressure and how it impacts the data freshness ingested into the cloud layer.

As mentioned in the implementation details, the setup consists of 2 sensor emulators emulating three sensors each for this experiment. These sensor emulators are connected to two edge devices with a balanced configuration where each edge device receives three sensor streams. Both sensor emulators and edge devices are implemented on Raspberry Pi devices. All the devices are connected to the network using wired LAN cables through a 1 Gigabit LAN switch. A router is connected to the switch, providing edge layer access to the cloud layer as shown in Fig 3.5.

3.7.1 Performance Baseline

The processing load on the edge device is the prepossessing and forwarding overhead. This experiment establishes the baseline performance of the edge device for different data rates for the fixed load. Data rates are incrementally set on the IoT devices so that the total data rates are varied from 10 records per second to 450 records per second for each sensor. ie values are set for λ_{D_i} such that $\sum_{k\varepsilon G_i} \lambda_{D_i}$ is varied from 100 to 4500 records per second. The effective forward rate is measured for each data rate for a fixed period. The results show that

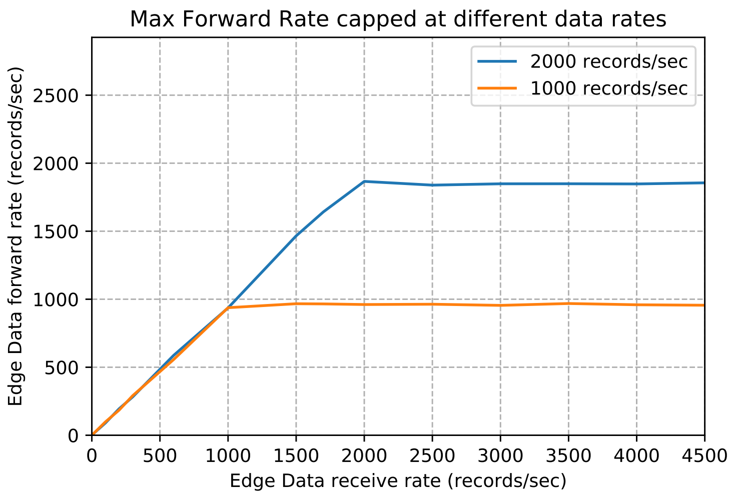
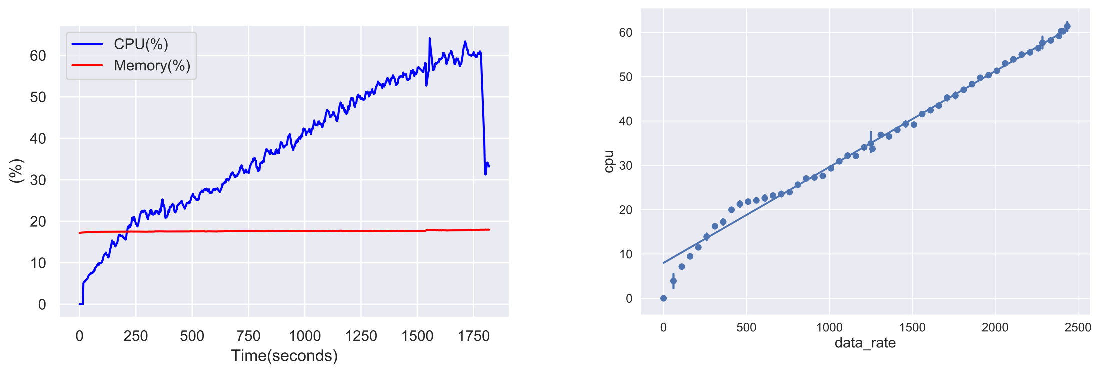


Fig. 3.6 Results of baseline performance in terms of forwarding rates for different data arrival rate

the forward rate matches the receive rate until capped forward rates (1000, 2000 records/sec), after which backpressure started building and affecting the forward data performance, making the forward rate dip. Another test is done by increasing the data rate until the back pressure starts building. This will show the CPU and memory characteristics when performing normally without back pressure.

3.7.2 Backpressure Detection

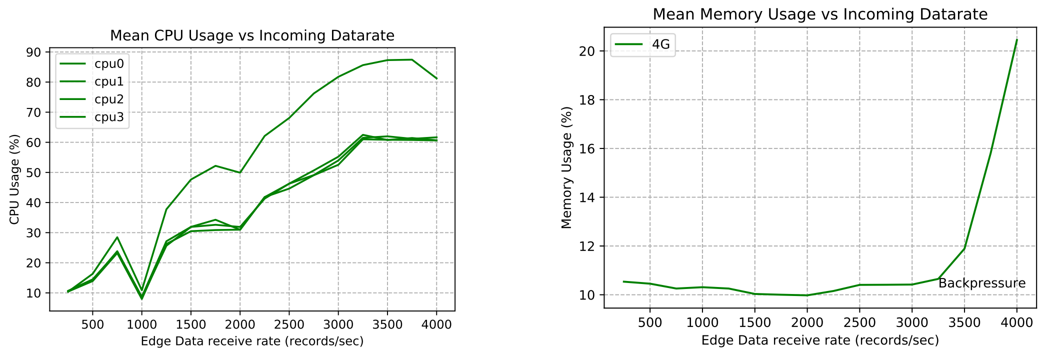
In this test, we try to identify the point at which the system reaches the state where the backpressure in the edge device starts building. Each edge device is set for this with a max forward rate of 5000 records per second. The data rates of sensors are elevated gradually to identify the point at which the edge cannot forward all the data received. CPU and memory



(a) CPU and Memory (b) Data Rate - CPU Usage Regression Line

Fig. 3.7 Results of baseline performance under normal operation without back pressure

stats of the edge device are monitored and plotted against the data receive rate. Results (Figure 3.8) show the point at which the backpressure of the data spikes up the resource utilisation, especially the memory utilisation. Another test is conducted without a fixed max forward rate which may lead to CPU/Memory saturation. This experiment helps to identify the point at which CPU/Memory saturates, leading to backpressure and, eventually, the unstable behaviour of the edge device. The results show the total CPU saturation point around 63% and the rapid spike in the memory when the CPU is saturated.



(a) CPU usage for different data rates (b) Memory usage for different data rates

Fig. 3.8 CPU and memory usage of edge device for different data rates

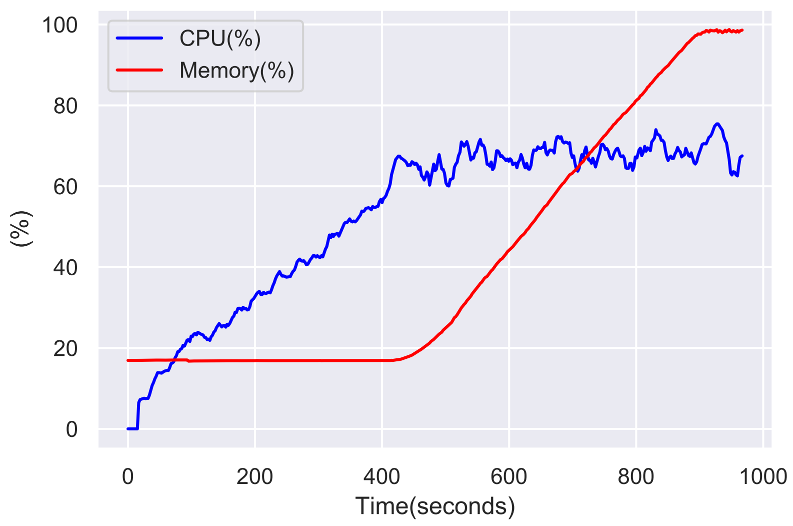


Fig. 3.9 Data Rate - CPU and Memory: Linear Ramp up test until the device crashed

3.7.3 Data Freshness

The clocks of all the devices are synchronised using a standard Network Time Protocol (NTP) server. Each record produced in the sensor simulator is attached with an epoch time. These records are sent to the cloud through edge devices. The edge device attaches its current timestamp to the record before forwarding it to the cloud. This timestamp attached by the sensor is compared with the epoch time in the cloud environment to measure the latency/data freshness. The latency is tested with three different network scenarios: 4G, 3G and GPRS. The results show that the latency is showing a sharp spike at the point of the forward rate limit.

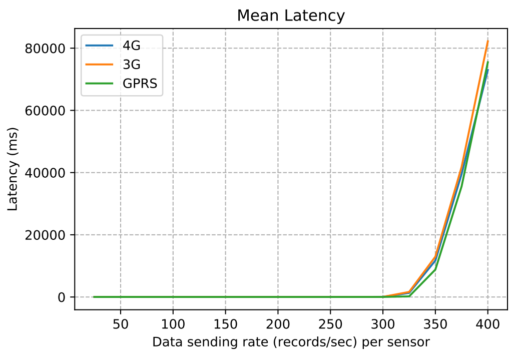


Fig. 3.10 Results of latency test for 4G, 3G and GPRS network

3.8 Discussion

Here, we discuss the outcomes of the tests conducted to assess the Emulation Toolkit. The three main topics of discussion are data flow rates, the onset of backpressure at the edge device, and how adaptive flow management handles backpressure and its effects on data freshness.

3.8.1 Data Flow Rate

In the first set of experiments, baseline performance (Section 3.7.1) is identified. For this, the data rate from the sensors to the edge devices is increased linearly at a rate of 200 records per second. Results show that there is a linear relation between the incoming data rate (Figure 3.7) and the CPU usage until the point where the CPU is not capable of processing more records. In the second set of experiments, we tried to detect the back pressure (Section 3.7.2) by performing the linear ramp up until the edge device became unstable and crashed. This is to capture the point where the back pressure starts building and eventually making the edge device unstable. Results show that for RPi 3B+, the CPU load went up to 63% and stayed under 75% for a long time and eventually crashing the system (Figure 3.8). The system functionality i) and ii) are evaluated by checking the monitoring results and how they match the edge device's actual system stats. A caveat of this approach is that the CPU usage percentage was calculated for all the cores, while some cores were utilised more than others. In this case, some CPU cores were the bottleneck while the total CPU usage across all the cores remained under 63%. The CPU threshold detection model could be changed to consider the maximum usage across all the CPUs. The most evident and trivial reason for back pressure is that the incoming data rate is more than the outgoing data rate. Apart from edge device processing capacity and memory, data flow is also affected by network bandwidth latency and jitter. In these experiments, the networking factors are not considered, and network bandwidth and latency are kept relatively stable with LAN networking. The messaging broker in the edge device holds the data in a queue until it is consumed for processing. Memory usage is observed to be growing with the increase in the queue length.

(Section3.7.1) shows the edge device's capacity to transmit data at various data flow rates. Figure 3.6 illustrates how the forward rate and receive rate coincide up to a certain point, at which point backpressure begins to accumulate and influence the forward data performance. The edge device can manage data up to a particular data flow rate before its performance starts to degrade.

Additionally, Figure 3.7 displays the CPU and memory use in a backpressure-free environment during typical operation. The findings demonstrate a distinct relationship between data rate and CPU consumption (Figure 3.7b). This information is vital for understanding the system's limitations and planning for efficient resource management on edge devices.

3.8.2 Backpressure Detection

The backpressure detection experiment (Section 3.7.2) identifies the point at which the edge device begins to experience backpressure. It is evident from Figure 3.8 that the system's memory utilisation swiftly increases, indicating the onset of back pressure. This information can be used to determine the data handling capacity and resource utilisation limitations of the edge device.

In addition, the experiment without a fixed maximum forwarding rate (Figure **??**) demonstrates that the edge device becomes unstable when the CPU reaches approximately 63% saturation, followed by a rapid increase in memory utilisation. This finding is essential for establishing secure operating limits for edge devices and preventing potential failures or instability.

3.8.3 Data Freshness

The data freshness experiment (Section 3.7.3) examines latency or data freshness in various network scenarios (4G, 3G, and GPRS). Figure 3.10 demonstrates a significant increase in latency at the point of the forward rate limit. This observation suggests that the adaptive flow control mechanism should be designed to resolve this issue and maintain an acceptable level of data freshness in the face of backpressure.

Chapter 4

Coordinated Data Flow Control in IoT Networks

4.1 Introduction

The edge and cloud solutions integration has been effectively implemented in numerous cutting-edge systems [20, 116]. These systems focus on relocating analytical tasks closer to the data source, thus reducing latency. The ApproxIoT system [139] was developed to harness edge computing resources, such as mobile phones, network gateways, and edge data centres at ISPs, for low latency analytics through approximate computing. However, IoT networks' inherent uncertainty impacts edge devices' stability and communication with the cloud. For instance, the available bandwidth of a cellular network can vary significantly due to fluctuations in the number of devices connected to a base station. Additionally, cellular networks in rural areas often switch dynamically between 2G, 3G, and 4G, where base stations are limited[31]. In many instances, high-frequency data is unnecessary when there is minimal variation in values or no immediate real-world situation demanding rapid data analysis. Factors such as sensor types, location, and real-world scenarios determine the data transmission rate or frequency. For example, a sensor measuring the ambient temperature of a factory floor may send readings twice per second. In contrast, a vibration sensor in a critical location of a compression pump may transmit data at a rate of 100 readings per second. Rapid response to significant changes in vibration is crucial, as any delay may lead to severe damage.

This work presents a data flow control system which can adapt the data forwarding from edge to cloud based on the uncertainty of the IoT network. To this end, we develop an *Edge Gatekeeper (EGK)* which monitors the status of each edge node while dynamically adapting the data ingestion rate from the IoT sensors to edge nodes and edge nodes to the cloud data centre. The design of the *EGK* allows the edge node to process and deliver only a subset of the data to reduce the latency.

Our proposed system needs to consider the following research questions:

- *What are the methodologies for achieving the maximum data flow rate in the IoT network utilising the available bandwidth while maintaining the stability of the network and the nodes?*
- *What are the methodologies for efficiently coordinating a large number of heterogeneous edge nodes and IoT devices sending data to a cloud simultaneously?*

To address the research questions mentioned above, this work makes the following contribution

- We propose a system and a reference architecture for IoT which coordinates the operations of edge devices to ensure stable and resilient operation using Dynamic Flow Allocation and a Coordinated data flow control mechanism. It is capable of coordinating a large number of heterogeneous edge nodes to utilise the computing resources in the cloud efficiently. We also provide a formal model of data flow in an IoT system.

- We validated our proposed models and the system architecture through a set of experiments using real-world data and a testbed. Our experimental results support the effectiveness of flow control for the stable operation on edge devices and resilience to the changes in the number of sensors and the data rate from the sensors.

Outline. The remaining chapter is organised as follows. Section 4.2 introduces the Data Flow Control methodologies. The proposed model and algorithms are explained in Section 4.3 followed by the evaluation by experimentation and discussion in Section 4.5.

4.2 Data Flow Control

The purpose of the flow controller is to actively control the rate of data being processed and forwarded from the edge device to the cloud. To reduce the amount of data being sent by the sensors, the edge gateway can reconfigure IoT sensors to set the data rate. This acts as a stabilisation mechanism for the varying workload at the edge layer. While designing the reference architecture (Figure 3.1), we considered the type, frequency and priority of the sensors. We considered the scenario of a single-hop star network where the sensor IoT devices are interacting directly with the edge device. It consists of 1) IoT layer, which contains the IoT devices/sensors having the lowest processing capacity, 2) Edge layer, which consists of devices which act as the entry points to the larger cloud network and are also capable of doing some processing of the sensor data, 3) Cloud layer with the highest processing capacity of all the other layers. Data flows from the IoT to the cloud layer through the edge layer. The data flow control is done using Edge GateKeeper (EGK), which is deployed in the edge layer and Adaptive Flow Controller (AFC) deployed in the cloud layer. This work proposes the system architecture for dynamically reconfiguring the edge device based on the number of devices connected and the priority of each sensor in the network.

Two methods are developed for managing the data flow. **i) Dynamic Flow Allocation:** The first method performs a flow allocation for each sensors based on the priority. When the edge device have a change in the total maximum data rate, this algorithm allocates the data rate on each sensors based on the priority of each sensor. **ii) Adaptive Data Rate Control:** The second algorithm allows the edge devices to set the maximum allowed data rate based on the network conditions by detecting the back pressure and resource utilisation of the device. Back pressure in the data flow pipeline is detected by observing the rate of change of the queue size in the messaging broker. Messaging broker queue length and CPU usage of the edge device are observed using APIs developed for the Edge Gatekeeper.

4.3 Model

Dynamic Flow Allocation and Adaptive Data Rate control algorithm work in tandem to manage the data flow between in the IoT network. This section explains the models supporting both Dynamic Flow Allocation and Adaptive Data Rate Control.

4.3.1 Dynamic Flow Allocation

Assume that we have a set of IoT devices D, and each one $D_i \in D$ has its priority $\mathscr{P}_i$ and sending rate λ_i. These devices are partitioned into K groups, and each group $G_k, k \in K$ interacts with only one edge node E_j. As a result, the total number of edge node $|E|$ equals the number of groups K, where E is a set of edge nodes. Finally, all edge nodes forward the received data streams to the cloud data centre

We define the data flow as a tuple $\langle \mu_j, \lambda_i \rangle$, where μ_j is the forwarding rate of E_j while λ_i represents the data injection rate from D_i to E_j. We consider the situation that IoT devices are generating more data streams than the edge node can process and forward. This situation can occur because of two main reasons, i)Edge does not have enough resources to process

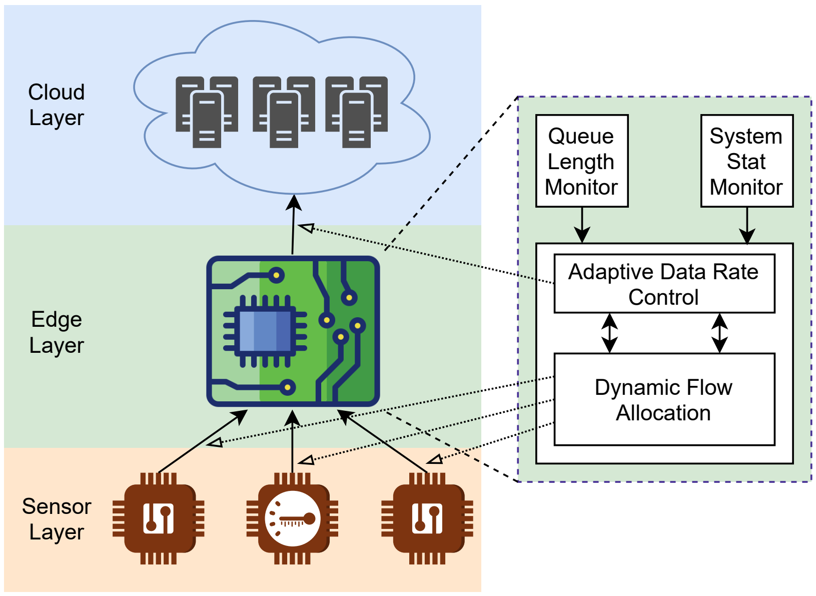


Fig. 4.1 Adaptive Datarate Control and Dynamic Data Rate Control

and forward the amount of data it receives ii) Edge node has assigned a maximum data forwarding rate it could perform to the cloud when the total incoming data rate is higher than the forward rate. For example, the injection rate of a group of IoT devices G_j which are connecting with edge node E_j is larger than the forwarding rate of E_j, noting μ_j, i.e., $\sum_{i \in G_j} \lambda_i > \mu_j$.

To overcome this, we design a flow control method that prioritises the data streams which are coming from high priority IoT devices by dynamically reducing the injection rates of the less critical IoT devices. We set weights to each sensor according to the priority. Possible values of priority $\mathscr{P}$ are {HIGH, MEDIUM, LOW}. Weights W corresponding to these priorities are $\{3, 2, 1\}$. Weights associated with IoT device D_i is represented by W_i.

The data rate allocation for device D_i is given by

$$A_i = \frac{W_i}{\sum_{k \in G_j} W_k} \times \mu \tag{4.1}$$

Table 4.1 A summary of symbols and abbreviations used within the formal definition.

Symbol	Explanation
D_i	IoT Device
$\mathscr{P}_i$	Priority of IoT device D_i
K	Number of partitions of IoT devices
G_k	Group of IoT devices
E_j	Edge device
$\|E\|$	Total number of edge nodes
λ_i	Data emission rate of i^{th} IoT device
S_i	Data size of each value of i^{th} IoT device
N	Number of IoT devices
K	Number of Partitions of IoT devices
$\langle \mu_j, \lambda_i \rangle$	Data Flow
μ_j	Forwarding rate of Edge node E_j
λ_j	Data injection rate from D_i to E_j
W_i	Weight given to the IoT device D_i
$A[n][m]$	Data rate allocation matix. Rows represent edge devices and columns represent IoT devices

Algorithm 1 finds the data rate of individual IoT devices. Data stream ds for each IoT device is again a tuple $ds = \langle \lambda_i, S_i \rangle$ where, λ_i and S_i are the data rate and size of data generated by the IoT device D_i. We therefore define the dataflow mapping as a tuple $\langle E_j, \mathscr{C}, \lambda_{\{D_i \to E_j\}}, \lambda_{\{E_j \to C\}} \rangle$, where the $\lambda_{\{D_i \to E_j\}}$ represents the data rate from device D_i to edge E_j and $\lambda_{\{E_j \to C\}}$ represents the data rate from edge E_j to cloud data center C. Consider there are X number of edge devices each represented by the tuple $\langle id, \mathscr{C}, \lambda_{\{D_i \to E_j\}}, \lambda_{\{E_j \to C\}} \rangle$ where id is the identifier of the edge device, $\mathscr{C}$ is the capacity of the edge device which is again a tuple $\mathscr{C} = \langle \mathscr{R}_S, \mathscr{R}_H \rangle$ where $\mathscr{R}_H$ and $\mathscr{R}_S$ are the hardware and software support provided by the edge device. Similarly, cloud datacentre C is represented as a set of components (VMs or containers) with an incoming data rate constraint of λ_C.

Based on the given priority of IoT devices D_i, the edge-to-cloud communication is sampled according to the function $\mathscr{F} : \lambda_{\{D_i \to E_j\}} \to \lambda_{\{E_j \to C\}}$. To perform this operation, the edge needs to compute the data rate for each IoT sensor device, which enables the sampling of the data.

Problem Formulation: Given the IoT infrastructure, D, E, C,

$$\text{maximize}\{\lambda_{\{E_1 \rightarrow C\}}, \lambda_{\{E_2 \rightarrow C\}}, ..., \lambda_{\{E_K \rightarrow C\}}\}$$

with constraint to:

$$\sum\{\lambda_{\{E_1 \rightarrow C\}} + \lambda_{\{E_2 \rightarrow C\}} + ... + \lambda_{\{E_K \rightarrow C\}}\} \leq \lambda C \tag{4.2}$$

$$\exists_i \exists_j \{\sum_i \lambda_{D_i} \leq \lambda_{E_j}\} \tag{4.3}$$

where there is a mapping between D_i and E_j

Constraint 2 explains that the data rate of all the edge to cloud communication should be less than the available data rate of the cloud. Similarly, Constraint 3 explains the data rate limitation of each edge device E_j.

Algorithm 1: Data Rate Controller Algorithm

```
Input  : $A[n][m]$: A 2-D matrix where rows represent Edge devices and columns
         represent IoT Devices; Values represent the data rate from the IoT device to
         the Edge device; n edge device; m IoT devices
$\mathscr{P}[n][m]$: List of Priority for each Edge device; n=number of Edge Devices
$\lambda$: Total data rate of all IoT devices connected to an Edge device
$\mu_j$: Data rate from edge to the cloud
Output : Data rate allocation matrix A[n][m]
1  Initiate weight matrix W[n][m]
2  Initiate total weight array $tw[n]$ with 0
3  for $i$=0 to n-1 do
4      for $j$=0 to m-1 do
5          if $\mathscr{P}[i][j]$='HIGH' then
6              $W[i][j]$=3
7              $tw[i] = tw[i] + 3$
8          end
9          else if $\mathscr{P}[i]$='MEDIUM' then
10             $W[i][j]$=2
11             $tw[i] = tw[i] + 2$
12         end
13         else
14             $W[i][j]$=1
15             $tw[i] = tw[i] + 1$
16         end
17     end
18 end
19 Initiate data rate allocation matrix A[n][m]
20 for $i$=0 to n-1 do
21     for $j$=0 to m-1 do
22         $A[i][j] = \frac{W[i][j]}{tw[i]} \times \mu$
23     end
24 end
```

4.3.2 Adaptive Data-Rate Control

In an IoT infrastructure, different types of networking and protocols are employed. Mostly the wireless technologies used in the IoT network varies in terms of the network bandwidth, latency and gitter. Edge devices connect to the cloud servers over wired and wireless connections. WiFi and celluar (2G/3G/4G/5G) are the most widely used types of connectivity for edge devices to the cloud infrastructure. Several factors, like temperature and humidity, influence the bandwidth and latency of the network in such scenarios. In many scenarios, the edge device may have to switch the network types based on the network conditions. For instance, it may switch from 4G to 3G when 4G signals are week, and this affects the bandwidth and latency of the network. Adaptive Data-Rate control(ADC) efficiently uses the maximum available data rate by monitoring the back pressure and the resource utilisation. ADC involves detecting the back pressure and mitigating it by changing the data rate according to the varying level of back pressure. This feedback loop-based data rate control ensures the network traffic's stability.

Detecting Back Pressure

Back pressure, in the conventional sense, is the resistance or force against the desired flow of liquid through a pipe. In a data pipeline, the flow of data behaves like a flowing fluid. In a data pipeline, the input is generally referred to as the data source and the destination of the data is referred to as the data sink. Like in fluid dynamics, if the rate of data ingested into the pipeline should match the amount of data going out of the pipeline. A back pressure develops when the rate of ingestion from the source exceeds the rate of output to the sink. In any data pipeline, a varying amount of back pressure occurs due to several factors like variability in the data processing speed, spikes in data rate and a dip in the rate of data sink. In an IoT network the pipeline of data is IoT sensors -> Edge -> Cloud. Changes in network bandwidth or faults may create back pressure in an IoT data pipeline. In any network, buffers

Table 4.2 Symbols used in the ADC Flow Chart

Symbol	Explanation
D_C	Current Data Rate
D_N	New Data Rate
D_{LS}	Last Stable Data Rate
D_{LU}	Last Unstable Data Rate
C_C	CPU Usage
C_T	CPU Usage Threshold
Q_L	Queue Length
Q_R	Rate of Change of Queue Length
U_{RU}	Linear Ramp Up Unit
U_{RD}	Linear Ramp Down Unit

are used to handle the variability in the data flow. In a three-layer IoT network, the buffering is handled by the messaging broker in the edge layer. Back pressure leads to a delay in the processing of data records on time. In a time-sensitive IoT network, delays in processing sensor reading and decision-making could have bad implications.

Back pressure in the data pipeline is detected by monitoring the queue size in the messaging broker. Queue size in the messaging broker does not essentially point to a problem in the data pipeline. However, a growing queue size indicates a condition where the downstream processors cannot process the data as much as it receive. We developed a back pressure detection mechanism for our Edge GateKeeper by monitoring the *rate of change* of the MQTT broker message queue length. Queue length measurement samples are taken every second, and the rate of change of queue length is measured by taking the average of deltas for a window of 10 seconds.

Data Rate Control

The data rate control work with the back pressure detection mechanism to decide the data rates to be set. Two methods have been used to detect the back pressure. The first method relies on the queue length of the MQTT broker and its rate of change. The second approach considers the CPU utilisation of the edge node as well as the queue length and its rate of change. Data rate control employs an adaptive multiplicative increase and multiplicative decrease algorithm. The data rate is increased and decreased based on the back pressure condition, the last know stable data rate. Figure 4.2 explains the algorithm flow chart.

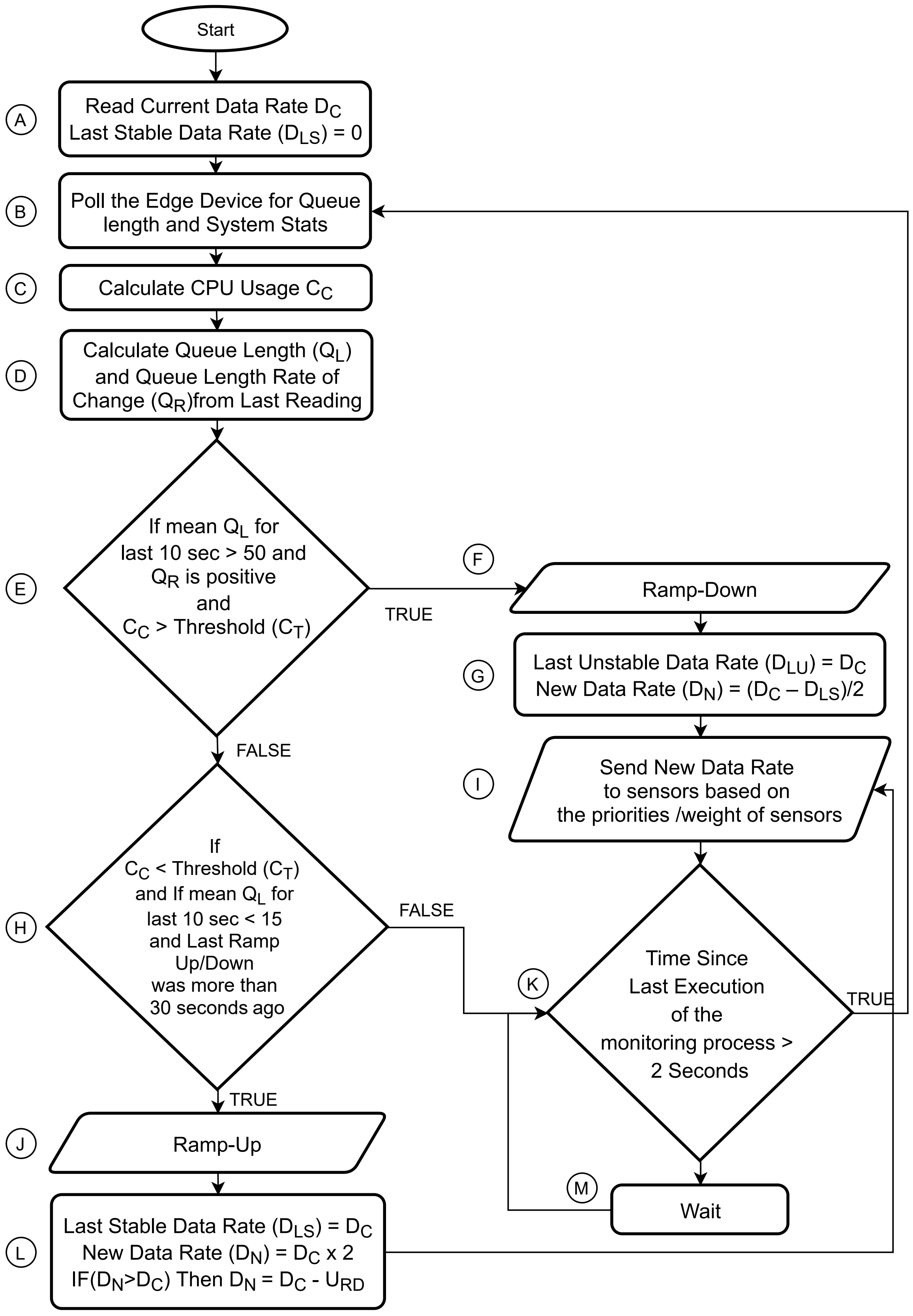

Start
A
Read Current Data Rate D_C
Last Stable Data Rate (D_{LS}) = 0
B
Poll the Edge Device for Queue length and System Stats
C
Calculate CPU Usage C_C
D
Calculate Queue Length (Q_L) and Queue Length Rate of Change (Q_R)from Last Reading
E
If mean Q_L for last 10 sec > 50 and Q_R is positive and C_C > Threshold (C_T)
TRUE
F
Ramp-Down
G
Last Unstable Data Rate (D_{LU}) = D_C
New Data Rate (D_N) = (D_C – D_{LS})/2
I
Send New Data Rate to sensors based on the priorities /weight of sensors
FALSE
H
If C_C < Threshold (C_T) and If mean Q_L for last 10 sec < 15 and Last Ramp Up/Down was more than 30 seconds ago
FALSE
K
Time Since Last Execution of the monitoring process > 2 Seconds
TRUE
TRUE
J
Ramp-Up
M
Wait
L
Last Stable Data Rate (D_{LS}) = D_C
New Data Rate (D_N) = D_C x 2
IF(D_N>D_C) Then D_N = D_C - U_{RD}

Fig. 4.2 ADC Flow Chart

4.4 Evaluation

This section explains the evaluation of the Dynamic Flow Allocation and Adaptive Data Rate Control strategies. To validate this system, we designed four experiments to run on the prototype. The primary objective of the experiments is to understand the impact of a different rate of data flow for a fixed load on the edge device. The second objective is to understand the point at which the edge device is rendered into an unstable state, where data flow backpressure starts building on the edge device. The third objective is understanding how adaptive flow control deals with the backpressure and how it impacts the data freshness ingested into the cloud layer.

4.4.1 Experimental Setup

The experimental setting for determining how well the Dynamic Flow Allocation (DFA) approach performs is described in this section. The experimental setup is intended to replicate different data rates and network latency, simulating actual IoT scenarios. The experimental settings consist of 2 sensor emulators emulating ten sensors each and Edge devices implemented on Raspberry Pi 3B devices, simulating a real-world IoT network. The sensors are connected to edge devices, which in turn are connected to a cloud server through 4G internet. The edge devices implement the Adaptive Data Rate Control to manage the data flow between the sensors and the cloud server. 3G and GPRS networks are emulated using Linux TC[15] tool.

Hardware Setup

The experimental hardware configuration comprises the following components.

1. Sensor Emulators: A collection of Raspberry Pi devices are utilised to simulate Internet of Things (IoT) sensors. These devices produce data at different intervals and send it to the EdgeGateKeeper.

2. Edge Gateway: The Edge Gateway, which is a Raspberry Pi device, is tasked with the responsibility of receiving data from the Sensor Emulators and transmitting it to the Cloud Instances. EdgeGateKeeper software is installed on these devices

3. Cloud Instances: A number of cloud instances (e.g., AWS EC2 instances) are used to simulate the cloud layer in the IoT environment. Each instance is configured with a specific processing capacity and network bandwidth.

Software Setup

The software components for the experiments include:

1. MQTT Broker: An MQTT broker is installed on the Edge Gateway to facilitate data transmission between the Sensor Emulators and the Cloud Instances.

2. EdgeGateKeeper: The EdgeGateKeeper software is installed in the Edge Gateway Raspberry Pi's

3. Data Processing and Storage: Each Cloud Instance is equipped with data processing and storage services to process and store the received data. Cloud

4. DFA Algorithm Implementation: The DFA algorithm is implemented within the Cloud layer to allocate data rates for sensors dynamically.

4.4.2 Adaptive Data Rate Control

In order to test the efficiency of the ADC, back pressure recovery time is measured. The table (Table 4.3) shows the recovery time from the backpressure for each data rate when AFC is engaged.

Table 4.3 Back pressure recovery time with AFC

Data Rate(rec/sec)	Recovery Time (seconds)
3250	∼2
3500	∼2.5
3750	∼4
4000	∼6
4500	∼7

Adaptive Data Rate Control is turned on, and the system is allowed to adjust the data rate based on the queue length. The experiment setup consists of a sensor emulator emulating three sensors, which is connected to a edge. Both sensor emulators and edge devices are implemented on Raspberry Pi devices. We used a sample of data for 1 week from different sensors in the Newcastle urban observatory with a mean record length of 75 characters/75 bytes. For this experiment, the length of the queue in the messaging broker is also monitored. The change in queueing and memory is calculated every 2 seconds. The recovery time is calculated as the time it takes for the messaging queue to be empty, and the change in queue size is stable around 0. Two variants of ADC algorithm is tested and compared. One is based on the CPU usage and queue length, while the second algorithm relies only on the queue length.

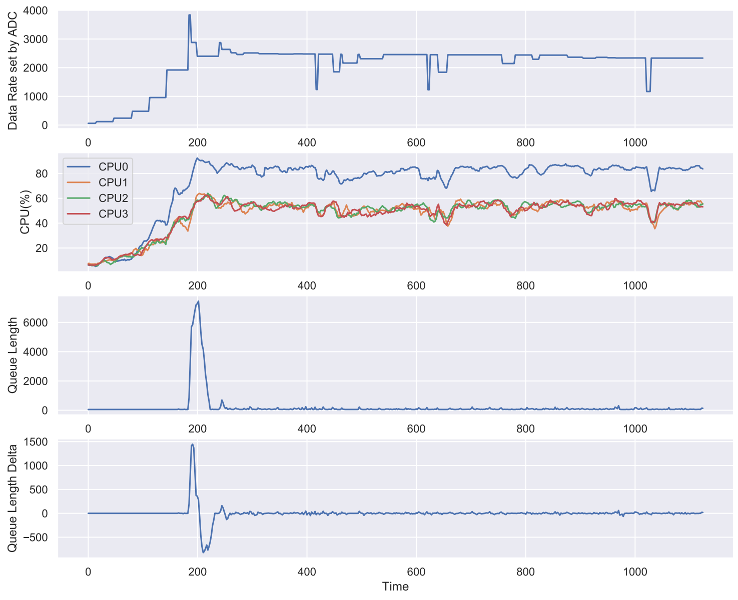


Fig. 4.3 ADC adjusting data rates based on Queue lenght

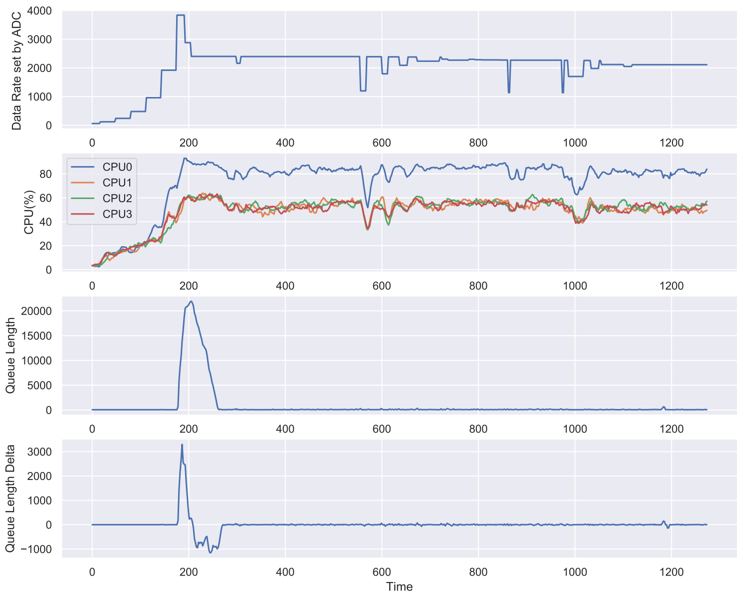


Fig. 4.4 ADC adjusting data rates based on CPU usage and Queue lenght

4.4.3 Dynamic Flow Allocation

To validate the Dynamic Flow Allocation (DFA) approach the sensor data received at the cloud layer is analysed. The main objectives of these experiments are to assess the capability of the DFA in balancing the data flow among multiple sensors, ensuring optimal resource utilisation, and maintaining data freshness in the cloud layer. This evaluation verifies the effectiveness of Dynamic Flow Allocation in setting the data rate for individual sensors based on the priority set for each sensor.

Out of the 20 sensors, 5 of them are given highest priority and 8 medium priority and 7 the lowest priority. DFA allocated the total available bandwidth based on the priority of the sensor. To evaluate this, the data received at the cloud server is analysed for the distribution of data by sensors.

Consider the total data rate available for allocation is D_{total}, we can distribute the data rate among the sensors according to their priority levels. Let the data rate allocated to the highest, medium, and lowest priority sensors be D_{high}, D_{medium}, and D_{low}, respectively.

We assign weights to each priority level as 3 for the highest priority, 2 for medium priority, and 1 for the lowest priority. The sum of weights will be:

Sum Weights = (3 x 5) + (2 x 8) + (1 x 7) = 15 + 16 + 7 = 38

Then, the proportion of the total data rate allocated to each priority level:

$$D_{\text{high}} = \frac{3 \times 5}{38} \times D_{\text{total}} \tag{4.4}$$

$$D_{\text{medium}} = \frac{2 \times 8}{38} \times D_{\text{total}} \tag{4.5}$$

$$D_{\text{low}} = \frac{1 \times 7}{38} \times D_{\text{total}} \tag{4.6}$$

Suppose the DFA is allocating the data rate correctly. In that case, the proportion of data points/records received by the cloud server by each of these priority levels should be equal to the theoretical data rate proportion as mentioned in the above equations. Analysing the number of records for sensors in each priority level, the following are the results.

Table 4.4 Expected and actual record counts for each priority level

	D_high	D_medium	D_low
Expected Records	29,934,688 (39.47%)	31,930,333 (42.11%)	13,969,521 (18.42%)
Actual Records	32,792,452 (43.24%)	31,491,291 (41.53%)	11,550,799 (15.23%)

4.5 Discussion

The study aimed to assess the efficacy of Dynamic Flow Allocation (DFA) and Adaptive Data Rate Control (ADC) techniques in the management of data flow within the Internet of Things (IoT) systems. This section analyses the essential findings and potential improvements of the methodologies.

4.5.1 Adaptive Data Rate Control

Two variants of Adaptive Data Rate Control (ADC) are evaluated in Section 4.4.2. In the first run (Fig 4.3). Adaptive Data Rate Controller is enabled, and it can adjust the data rate according to the allowed bandwidth using i) queue length and its ii) rate of change. Based on these two parameters, the algorithm triggered the ramp-up and ramp-down of the data rate. Results show that the data rate is adjusted so that the queue length is stable and the CPU is not overloaded. The second run (Figure 4.4) shows the ADC's performance based on CPU load and queue length together. Ramp-down event is triggered when the CPU usage crosses a threshold (63% in this case). Results show that the data rate is adjusted to a stable point so the CPU is not overloaded. The response time of queue-based ADC is comparatively faster as queue length changes are detected instantaneously than CPU usage. The queue length-based algorithm performed more data rate changes, while the CPU usage-based algorithm reached stable data more efficiently. In the experiment based on CPU usage, the ADC performed 16 and 12 data rate changes, respectively. However, the queue length-based ADS performed around 24 data rate changes while the network and workload were identical. Considering network bandwidth, the queue length-based model is more reliable as the CPU usage is well under the threshold while the bandwidth is the bottleneck. In this experiment, we set the CPU threshold to 63% based on the previous observation of the saturation point. Determining the CPU usage saturation point ahead of time is a challenge. In the baseline performance test, we could find a linear relationship between CPU usage and the data rate. However, these characteristics cannot be generalised for all workloads.

The backpressure problem in IoT systems was addressed by the use of the Adaptive Data Rate Control technique. The results of the trials show that the ADC approach can successfully deal with backpressure and modify the data rate based on the queue length (Figure 4.3) or a combination of CPU utilisation and queue length (Figure 4.4). This shows how the ADC technique can adjust to changing system circumstances and handle the data flow appropriately. There is room for development, though. The influence of various system resources like memory and network bandwidth may not be appropriately taken into account by the existing ADC technique. Other resource metrics could be integrated into a more thorough strategy, allowing the ADC strategy to better adapt to various IoT scenarios and offer even more effective data flow management.

4.5.2 Dynamic Flow Allocation

The goal of the Dynamic Flow Allocation technique was to allocate the available data rate among the sensors according to their relative priorities. The percentage of records received by each priority level provided by the assessment results demonstrated that the DFA algorithm was successful in achieving this objective. This highlights the DFA strategy's capacity to optimise resource usage, preserve data freshness in the cloud layer, and balance data flow among various sensors.

The expected and actual record counts for each priority level do not match up exactly, though (Table 4.4). These variations might be explained by a number of variables, including sensor data rate unpredictability, processing delays, and network latency. Additional research into these elements and how they affect DFA performance may help in improving the algorithm.

Additionally, the static priority assignment is the foundation of the current DFA algorithm. Sensor priorities may alter dynamically over time in actual IoT applications. Dynamic priority updates may be added to the DFA method in the future, enabling it to better adjust to shifting sensor priorities and offer better data rate allocation.

Chapter 5

Real-time Data Analysis Pipeline for Landslide Early Warning System

5.1 Introduction

A complex Internet of Things (IoT) environment comprises various components, including devices, networks, cloud infrastructure for storage and processing, analytics tools, applications, and security measures. The Internet of Everything (IoE) [52, 94] expands upon the IoT concept by integrating people and processes, enhancing early warning systems and smart city applications through the involvement of human sensors and improved prediction accuracy. Modern early warning systems and smart city applications often utilise social media feeds to extract contextual information about developing situations[42, 115]. IoE aims to create a more interconnected world where devices, data, and human interactions facilitate better decision-making and drive automation. A crucial requirement for IoE applications is the ability to perform real-time or near real-time[73] processing to produce actionable insights or enable automation.

Processing the continuous flow of information from distributed IoE sources at uneven rates in response to complex queries can be challenging. Data Stream Processing (DSP) engines have been implemented in numerous domains like smart cities to address these challenges[130]. Modern DSPs[61] are highly scalable and perform stream analysis efficiently, but the question remains whether they can scale beyond a cluster of DSP engines. As the IoT and IoE continue to grow as highly dispersed data sources, more distributed processing across heterogeneous platforms is necessary.

Existing DSPs distribute processing across a cluster of machines that are, for the most part, uniform in nature. This study explores a highly distributed "actor-based" [25, 8, 10] approach in which processing occurs on multiple platforms, with data made accessible via a distributed messaging system. Apache Kafka[72], a proven pub/sub-based[51] distributed event streaming engine, is used as a scalable data backbone for streaming and storing data from disparate sources, distributing the processing load according to a set of QoS parameters. This processing mechanism is fundamentally distinct from existing DSPs because processing is distributed and pipelined across multiple platforms/machines.

For instance, a smart city application for proactive air pollution control uses multiple data sources[63] to make traffic control and rerouting decisions, relying heavily on air pollution data streams and city road video streams. Numerous real-time air pollution inference models utilise proprietary software running on a specific platform, while video stream processing and inference require machines with high processing capacity and GPUs. Non-distributed DSPs cannot handle such workloads. In this context, we propose an architecture for a distributed stream processing system capable of performing complex processing on the stream and pipelining the data processing tasks.

For an Early Warning System (EWS) like a Landslide Early Warning System, sensor and social media data fusion can significantly improve precision. The methodologies described in this chapter address the following research questions:

1. How can a distributed real-time analytics pipeline be used for real-world applications like early warning systems and smart cities?

2. What methods can be used for pipelines to be efficiently monitored and scaled?

To address the research questions mentioned above, this chapter outlines the following research contributions:

1. We evaluated our Landslide Early Warning System using real-time social media data and real-world sensor data.

2. A system is designed to orchestrate streaming workflows distributed across multiple processing units on the edge and in the cloud.

The structure of this chapter provides an exploration of these research contributions and their potential implications. Section II delves into the history and related work of the Complex Event Processing (CEP)[35] and Event Stream Processing(ESP)[21] communities. The functional and technical architecture of the proposed processing model is described in Section 5.3. The Implementation and Evaluation are discussed in Section5.4 and Section 5.5.

5.2 Background

5.2.1 Complex Event Processing

Complex event processing (CEP), a class of information flow processing system views the flow of information as a notification of events[84]. These systems are capable of identifying complex patterns of events from multiple heterogeneous streams of data[47]. They have a wide range of applications, from control systems, financial services, and business process management to more recent IoT technologies[50]. CEP have a different approach based on traditional Context-Based Publish-Subscribe systems. Information comes into the system as messages coming from multiple publishers. The information sent out from the system are notifications to the interested subscribers, defined by detection rules. Complex event processing model is based on event-driven information systems. A complex event is defined as an event that could only happen if a lot of other events happened. For example, suppose it is raining heavily, and it is happening only because of the number of events in the atmosphere that lead to heavy rain. There exists a causal history for a complex event to happen. CEP systems are built to identify higher-level events from the complex series of lower-level events. Events relations are based on cause, by timing and by membership. This helps to analyse events to answer questions concerning higher-level events.

5.2.2 Data Stream Processing

Data stream processing is a data management approach in which data is handled in real time as its being generated. This approach is beneficial in scenarios where large volumes of data are produced continuously, such as financial transactions, market feeds, social media feeds and IoT sensor data. By processing continuous feeds of real-time or near-real-time data, decisions can be made quickly and acted on, which enhances efficiency. It enables the extraction of valuable insights that might be too vast for traditional batch-processing techniques. Various tools and frameworks have been developed to enable data stream processing with their own features and capabilities. Apache Kafka[72], Apache Flink[38] and Apache Samza[102] are some of examples of this. These technologies make stream processing pipelines highly scalable, fault-tolerant, and effective. Organisations can create pipelines that filter, aggregate, transform, and analyse data streams in real-time by utilising these frameworks, giving them the ability to react quickly to new trends, anomalies, or business opportunities. Data stream processing is now crucial to modern data architecture, enabling businesses to remain adaptable and competitive in a data-driven world.

Data Stream Management Systems (DSMS)[58] has its root in DBMS and approaches the problem as an evolution of traditional DBMS. It processes streams of data coming from different sources to produce new output data streams. Designed to process transient data, which is updated continuously. Executes standing queries which run continuously and provide results as and when new data arrives. DSMSs process data with transformations based on SQL operators like selection, aggregates, joins and other operators defined by relational algebra. Modern data stream processing engines[72, 38] like Apache Storm, Spark Streaming, and Flink are highly scalable and capable of processing millions of events per second and can be scaled up across large clusters of machines.

5.2.3 Stream Processing in Early Warning Systems

Stream processing, which enables real-time or near real-time data processing, is critical to the effectiveness of modern early warning systems [47]. Numerous data sources are utilised in early warning systems for landslides, earthquakes, and tsunamis, enabling rapid analysis and timely actions. Data streams from various sources, such as weather stations, remote sensing platforms, ground-based sensors and social media feeds, must be processed and analysed in these systems as they arrive to ensure that emerging threats are quickly identified and appropriate countermeasures are implemented.

Using stream processing technologies, early warning systems can effectively handle the massive amounts of data generated by these sources, process the data in parallel, and make decisions based on the most recent information available. The ability to process data in real-time is critical for accurate and timely hazard detection, allowing authorities to warn affected populations and implement mitigation measures as soon as possible. Stream processing enables continuous analysis and decision-making, significantly improving the dependability and responsiveness of early warning systems. Stream processing, as opposed to traditional batch processing methods, which may cause delays due to the need for data accumulation and scheduled processing, allows for continuous analysis and decision-making.

5.3 Methodology

This section explains the approach to designing, implementing, and evaluating a scalable data stream processing pipeline early warning systems, focusing on leveraging Apache Kafka for real-time data analysis in early warning systems. As a data backbone, Apache Kafka enables efficient data management and provides valuable monitoring and orchestration capabilities. The proposed methodology addresses the challenges of dealing with large amounts of streaming data while ensuring prompt hazard detection and response. This section explains the processes and techniques required to build a robust and efficient real-time data analysis pipeline, leveraging the power of Apache Kafka for stream processing and system monitoring. Figure 5.1 shows the high-level functional architecture of the Landslide Early Warning System.

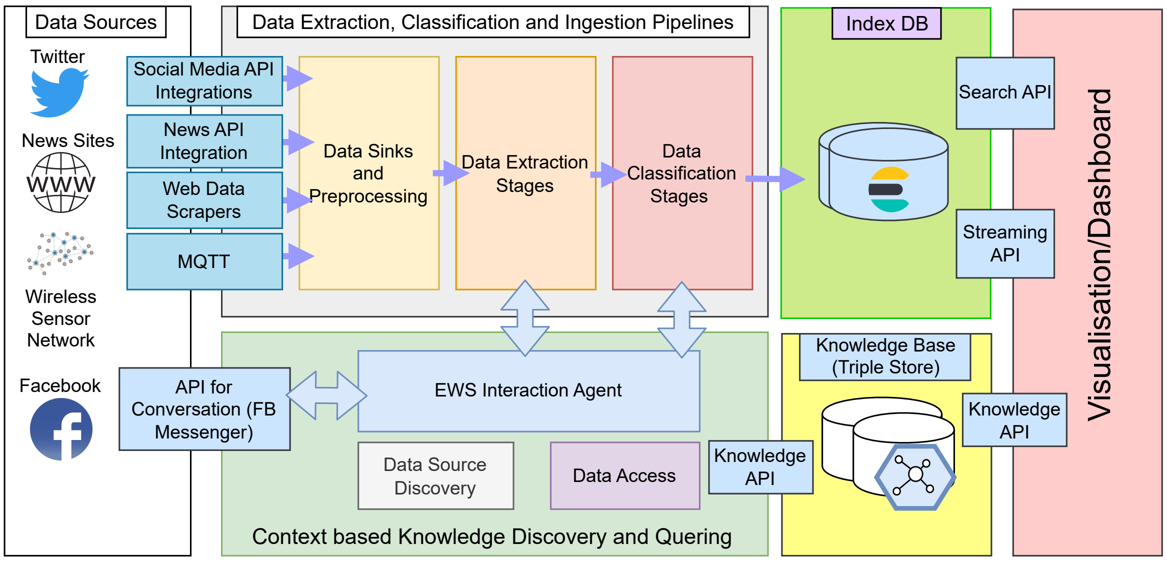


Fig. 5.1 High-level Functional Architecture

The key design considerations for the EWS based on social media, sensor network and weather data are

- Responsiveness to the events or the real-time processing capacity. i.e. the information should be processed, and useful information should be extracted and made available for the EWS operator in near real-time.
- Flexibility to add new data sources. The data model schema should be flexible to incorporate multiple data sources, like tweets and news articles
- Efficient information extraction, which includes spatial and temporal information.
- Indexing capability, which helps to search and retrieve data fast.
- Scalability and Reliability (Monitoring and Fault-Tolerance)

This chapter is mainly focusing on the orchestration and scalability of the processing infrastructure. The research is not focusing on the effectiveness of the detection of landslide events. Phengsuwan et al. (2019) discusses the Context based knowledge discovery for Landslip Early warning which deployed on the infrastructure this work proposes.

5.3.1 Steaming Data Pipeline: Concept

The principles of Flow-Based programming[97] and Actor model computation[62] are brought together in the concept of the Stream Processing Pipeline in order to produce a data processing pipeline that is both scalable and easily maintained. Streaming data pipeline is a flexible and modular solution for managing data flow in distributed systems that uses Apache Kafka as the data backbone and container-based applications as the processing units. Streaming data pipeline can be modelled as a directed graph consisting of nodes and edges. Edges signify information passing from one containerised component to another via Apache Kafka topics, and nodes stand for individual components.

Pipelining containerised applications is a method that divides a data processing system into smaller, self-sufficient parts, each of which executes a distinct task inside a container. Because these parts operate independently, multiple data items can be processed simultaneously at various points in the pipeline. Containers guarantee uniformity and processing-component isolation, while parallelism boosts throughput and efficiency. The integrity of the process is preserved as data moves in a predetermined order through the pipeline. The pipeline can deal with fluctuating data rates and recover from component failures with scalability, fault tolerance, and loose coupling provided by containerised applications.

The main features of this streaming data pipeline are

1. **Scalable Data Backbone**: Apache Kafka is the data backbone, allowing for reliable, low-latency streaming with high throughput. Kafka's distributed design makes it simple to increase or decrease the size of the pipeline in response to changes in the processing load or the amount of data being processed. This enables asynchronous communication between the processing components.

2. **Modular Processing**: To facilitate rapid deployment, scalability, and reusability, the stream processing pipeline employs containerised applications as processing units along the pipeline. Each module can be built, tested, and deployed separately from the others because it handles its own unique data processing function.

3. **Dynamic Data Flow**: This processing framework is based on Flow-Based Programming principles[97], which facilitates parallelism, concurrency, and asynchronous communication between components for effective data processing. The data is processed in real-time, guaranteeing a high throughput and minimising delays in the pipeline. It enables loose coupling and compostability of processing components.

4. **Observability and Management**: The data pipeline can be monitored and managed with Kafka's integrated monitoring and management system. This infrastructure enables monitoring of performance, data flow, and overall system health; it also permits configuration changes and the scaling of individual components.

5. **Extensibility and Interoperability**: The streaming pipeline is built to be easily extended with new features and technologies as they become available. Using a containerised approach, developers can run their code on any platform, regardless of the language or framework in which it was written. It also allows the containers to be deployed on specific platforms based on the requirement of the processing component. Deep learning model inferences run faster on GPU-accelerated machines where the containers with specific processing steps can be deployed. Hence, this methodology enables highly distributed but interoperable processing components.

In a containerised pipeline, Kafka topics are important in orchestrating data flow between processing components. Producers publish data to Kafka topics; consumers subscribe to these topics to read and process the data. When a component processes the data, it produces its output to another Kafka topic for the next component in the pipeline to consume. This sequential flow of data through Kafka topics ensures the integrity and order of the data processing while allowing for easy communication between the independently running components. Kafka's scalability and fault tolerance allows the pipeline to process massive amounts of data with minimal delays, making it an ideal data-flow mechanism for containerised software.

5.3.2 Data Ingestion

Data ingestion refers to the process of collecting and integrating data from diverse sources such as sensor networks and social media platforms, to enable processing and analysis in the subsequent stages of the pipeline. For Landslip early warning system, sensor data from the network of sensors deployed on landslide-prone areas, weather data from public APIs and social media feeds from Twitter were ingested.

Sensor Data Collection

Sensor data forms an integral part of an early warning system. Various types of sensors, including weather stations, soil moisture sensors, rain gauges, inclinometers, and extensometers, are deployed to gather real-time data on environmental parameters relevant to hazard detection. This data is ingested into the system using standard messaging protocols like MQTT. This is then fed into Kafka topics, which enables further processing.

Social Media Data Collection

Along with sensor data, social media platforms also provide real-time, location-specific contextual insights into ongoing events, making them valuable sources of information for early warning systems. Relevant Twitter feeds are collected based on predefined keywords using API provided by Twitter. This enables for the extraction of critical contextual information from social media feeds, which may aid in the early detection of hazards.

5.3.3 Data Processing

This section discusses the details of data processing steps and ML models and their execution workflow. The first step in processing is to use *User Classifiers* to identify who is posting this message. Currently, only two types of user classification are considered in this system: one is an official account, such as Meteorological Office; the other is for regular users. In general, the information provided by official accounts is more reliable than normal users. However, normal users can provide richer contextual information from the proximity of the hazard site. Since Twitter already provides user information, we use the Twitter handle to classify the users based on a dictionary. Next, the *Event Classifiers* identify whether a message relates to landslide hazard and antecedent hazard events (i.e., warning signs). These classifiers are developed by using spaCy, a NLP framework. To this end, a small amount of labelled data has been collected and then used this dataset to retrain spaCy's convolutional neural network (CNN) models [68, 111] to improve the accuracy of detecting events.

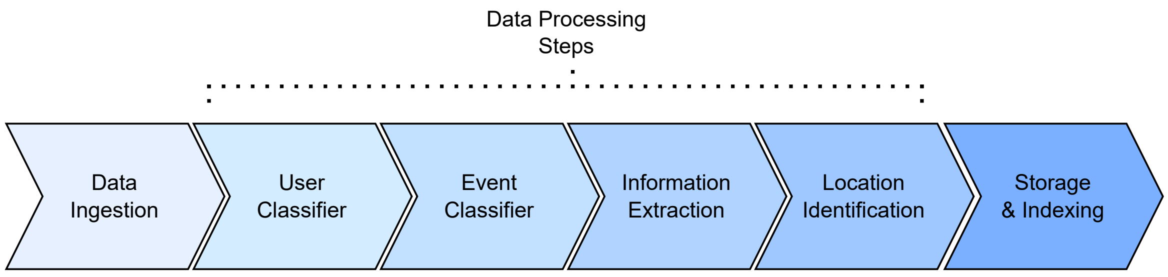


Fig. 5.2 Data processing steps

Obtaining the geolocation information from the mentions in the message is also essential for analysing landslides. This task consists of two steps: *Information Extraction* and *Location Identification*. The *Information Extraction* component aims to extract the named entities of a message using `en_core_web_lg` model (available on spaCy), a CNN model trained on OntoNotes[64]. These extracted named entities are the inputs of the *Location Identification* that classify the named entities as including geolocation information. If yes, these geo-names are converted to geo-coordinates by OpenStreetMap (OSM) datasets [1] using geocoding method[2]. A single geo-name may have multiple entries in the OSM dataset. The first entry from the OSM dataset is taken. In future works, the module will be seeded with the location of interest. Finally, all the outputs are published and stored in the Elastic Search Database for further analysis. More details about the data processing steps are explained in Appendix A.1

5.3.4 Data Storage and Indexing

For data storage and indexing Elasticsearch[59] is used as the primary data storage solution. Elasticsearch is a distributed, RESTful search and analytics engine built on Apache Lucene.

[1]https://wiki.openstreetmap.org/wiki/Downloading_data
[2]https://nominatim.org/

It supports geo-search and indexing for efficient geolocation-based querying and analysis, and it can handle large amounts of textual data and complex search queries.

Efficient full-text search operations like tokenising the text, filtering out common stop words, and applying text analysis techniques like stemming and lemmatisation are made possible using an inverted index for text indexing. Geolocation data is indexed using the GeoPoint data type, enabling location-based queries and analysis. Appropriate mappings and index settings are created to facilitate efficient querying and analysis. Elasticsearch stores the results of the Data Processing section's processing, such as user classifications, event classifications, extracted geolocations, and other valuable data. The system can quickly and accurately perform complex search queries, analyse the gathered data to detect and track landslip events and produce valuable insights and visual representations. Elasticsearch's fast and accurate search and analysis capabilities are made possible by the system's ability to handle large amounts of social media data through efficient indexing techniques.

5.4 Implementation Details

5.4.1 Kafka as Data Backbone

Kafka is a distributed system comprising servers and clients communicating through a high-performance TCP network protocol. It can be deployed on bare-metal hardware, virtual machines, and containers in on-premise and cloud environments.

Servers: Kafka operates as a cluster of one or more servers spanning multiple data centres or cloud regions. Some servers form the storage layer known as brokers. Kafka Connect[], continuously importing and exporting data as event streams to integrate Kafka with existing systems like relational databases and other Kafka clusters. A Kafka cluster is highly scalable and fault-tolerant to accommodate mission-critical use cases. If any server node fails, others will take over their tasks, ensuring continuous operations without data loss.

Clients: These enable the development of distributed applications and microservices that read, write, and process event streams in parallel, at scale, and in a fault-tolerant manner, even in the face of network problems or machine failures. Kafka includes some clients and supports dozens more provided by the Kafka community. Clients are available for various programming languages like Java, Scala, Go, Python, C/C++, and REST APIs.

Producers publish events to Kafka, while consumers subscribe to and process these events. In Kafka, producers and consumers are fully decoupled and agnostic of each other, a key design element for achieving high scalability. Kafka provides guarantees like processing events exactly once. Events are organised and durably stored in topics, which are partitioned and distributed across multiple Kafka brokers for scalability. This allows client applications to read and write data from/to many brokers simultaneously.

5.4.2 Pipeline Orchestration

Container Based Micro-services as Processing Modules

Microservice architecture organises an application as a group of services that can be independently deployed, tested, and maintained and which are loosely coupled to one another and centred on business capabilities. This architecture allows for the rapid and reliable delivery of large, complex applications by allowing individual services to scale and update independently. Containers provide a lightweight, standardised way for applications to move between environments and run independently. Except for the server's shared operating system, the code, runtime, system tools, libraries, and dependencies required to run the application are all contained within the container object. The pipeline processing components were deployed as containers which can be independently developed and tested. Containers can be placed on machines with specific hardware based on the processing components requirement, like GPU-accelerated node when the processing module performs a deep learning inference.

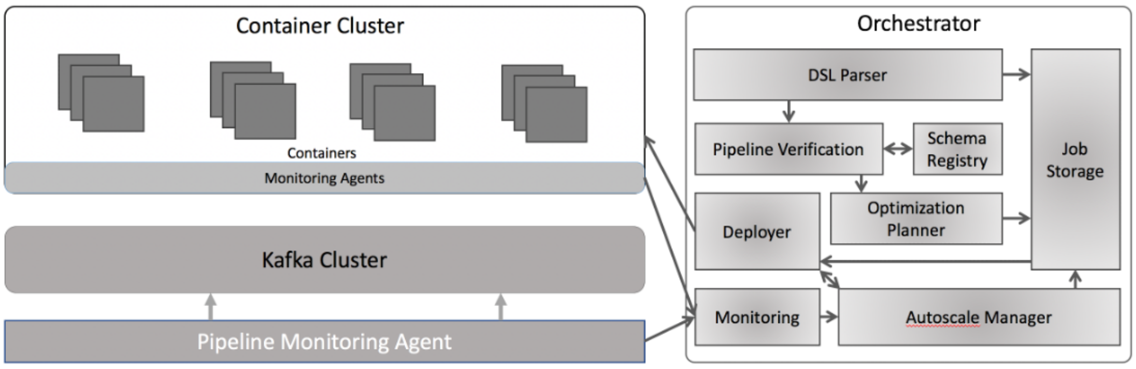


Fig. 5.3 Orchestrating Data Processing Pipeline

Deployment on Docker swarm Cluster

Each pipeline processing module is deployed as a container-based microservice. A Docker Swarm is a group of physical or virtual machines running the Docker application and configured to join together in a cluster. Swarm managers control the cluster's activities, while machines that have joined the cluster are called nodes. Docker Swarm is a container orchestration tool that manages multiple containers deployed across multiple host machines, providing high application availability. The container orchestrator ensures that each service's required number of instances runs and automatically starts new instances in case of failures.

Docker compose files are created with each of the processing components specified as a service. The input and output topic names are passed as environment variables. The services are deployed as a "docker stack" consisting of all the pipeline services.

Health Check and Monitoring

Service containers are monitored using cAdvisor, which tracks CPU usage, memory usage, and network in/out. Monitoring information is visualised using Grafana and focuses on key metrics like the number of messages in the pipeline and the number of messages processed by each step. The following tools are used for storing and visualising the monitoring information.

Prometheus: A open-source monitoring system that collects and stores metrics from the pipeline components. Prometheus provides a flexible query language for analysing collected metrics, enabling real-time monitoring and alerting.

Grafana: A visualisation tool that integrates with Prometheus to display the monitoring data in a user-friendly dashboard. Grafana offers customisable graphs and charts, making it easy to track the performance and health of the pipeline.

Consumer Lag Monitoring

One of the key metrics for monitoring the processing pipeline is consumer lag. This metric calculates the difference between the number of messages on the topic and the number of messages consumed by the consumer processing the message. Different processing steps in the pipeline may have different processing overheads. This may create bottlenecks in the pipeline. Bottlenecks in the pipeline can be identified by monitoring the consumer lag across each step. This enables system administrators to quickly identify performance issues, optimise pipeline components, and guarantee that the pipeline continues to perform as expected across a wide range of data rates. System developers can assess the pipeline's scalability by watching how it responds to an increase in data throughput thanks to consumer lag monitoring, ensuring the system can handle massive data streams in production. For early warning systems to function properly, it is essential to manage and optimise the performance of the pipeline that processes data streams in a timely manner, making consumer lag monitoring a vital tool.

5.4.3 Elastic Search as searchable Index

Elasticsearch[59] is a distributed, open-source search and analytics engine built on Apache Lucene[26, 83] and developed in Java. It began as a scalable version of the Lucene open-source search framework, with added horizontal scalability for Lucene indices. Elasticsearch enables storing, searching, and analysing large volumes of data quickly and in near real-time, returning answers in milliseconds. It achieves fast search responses by searching an index rather than the text directly. Elasticsearch uses a document-based structure instead of tables and schemas and offers extensive REST APIs for storing and searching data. Elasticsearch is a server that processes JSON requests and returns JSON data. The last step of the pipeline involves an Elasticsearch injector, which inserts incoming records into Elasticsearch as documents. This allows for efficient indexing and searching of the stored data, providing quick access to relevant information when needed. By utilising Elasticsearch, the system can accommodate large-scale data storage and retrieval, ensuring fast and reliable search responses even as data volumes grow.

5.4.4 Landslip Early Warning System Pipeline

Utilising Real-Time Data Pipelines, LEWS was created as a collection of containerised processing components communicating with individual Kafka topics. Sensor network data and social network(Twitter) feeds were the primary data types processed by the various pipeline components. Data is progressively analysed as data moves through the pipeline, and information is extracted and enriched. This data is added as metadata to the incoming data and is passed to the next stage for processing. As shown in Figure 5.4 the LEWs pipeline consists of 7 steps, each connected to the next step through Kafka topics.

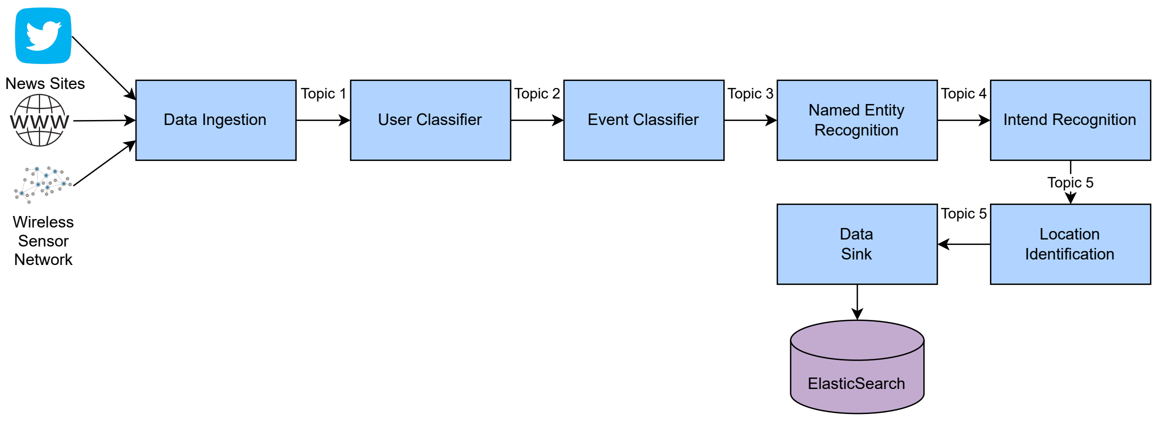


Fig. 5.4 Landslip Early Warning Stream Processing Pipeline

5.5 Evaluation

This section describes the evaluation of the proposed stream data pipeline system. The key objectives of this evaluation are to analyse the scalability, performance and resource utilisation under different workloads. The experiments were designed to measure the overall throughput, latency and resource utilisation while a high volume of data is streamed through the pipeline.

Since the Twitter streaming API only provides a certain number of Tweets per minute, we had to collect a sizable amount manually. The data collector module was fed with these pre-collected tweets to simulate high-velocity data flow. The average number of characters per tweet is 3330 bytes for the collected dataset, which roughly is around 3 KB per tweet. Because of the additional metadata and attributes added by Twitter, the size of the tweet object is larger. This metadata is important in the case of LEWS as it provides rich information about the users, like their signed-up location, and geolocation if it is enabled.

For the simplicity of experimentation only one data source is considered. However, adding more data sources to this system is relatively simple, considering the decoupled nature of the producing data to Kafka.

Experiment Setup

The experiment setup consists of a cluster of three machines with 16 GB of RAM and 4-core Intel Xeon processor. The following components

1. Kafka Cluster as data backbone by enabling data streaming between processing components
2. Elasticsearch as the data storage to store and enable fast searching using its indexes
3. Docker swarm as the container orchestration

On bare metal running Linux, a three-node Kafka cluster was installed. All three nodes now have the supporting service Zookeeper set up on them. Each machine had the Docker engine installed, making it a node in the Docker Swarm container orchestrator cluster. Since pipeline components are now considered services, they can be independently scaled. Using a Docker stack to deploy the services described in a YAML file as steps, the pipeline could be set up, run, and monitored with a single command. Kafka topics representing the input and output of each module were made available via environment variables.

For this experiment, a simplified version of the LEWs pipeline is used, which consists of the steps shown in Fig 5.5

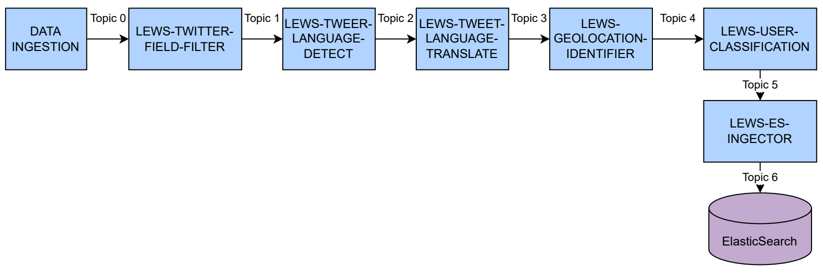


Fig. 5.5 LEWs Pipeline used for Experimentation

Throughput

Timestamps were appended to each record at the start and finish of the pipeline to calculate throughput and latency. NTP was used to ensure that all of the computers' clocks were within 1 millisecond of each other. Each record has a collection time stamp and a processing time stamp that were added in the first and second steps, respectively. However, the throughput of the pipeline is always dependent on the workloads in and the scale of the processing containers.

Throughput is the total number of records which pass through the pipeline per unit time. A set of 399,083 tweets are sent to the pipeline, and the time it took from the start of the pipeline to ingesting to ElasticSearch after all the processing steps were calculated. This experiment is repeated 5 times to take the average throughput.

Results

No. of Records/Tweets	Time (s)	Records/second
399083	2624	152.09
399083	2877	138.71
399083	2585	154.38
399083	2553	156.32
399083	2627	151.92
Average Throughput		150.41

Table 5.1 Pipeline Throughput

After ingesting the same dataset 5 times, results show an average throughput of 150.41 records/second.

Bottleneck Detection Using Consumer Lag

As mentioned earlier in the chapter, in Kafka, consumer lag metrics measure the time elapsed between the offset of the most recently produced message and the offset of the most recently consumed message for a given set of consumers. This is the total number of messages that have been received but have yet to be processed. Consumer lag measures how slowly recipients are processing transmissions from senders. Lower consumer lag indicates that it is keeping pace with the producers, if it is large, it could mean that the recipient's message processing is lagging behind.

In this experiment, data is published to the data ingestion process at a very high rate. This remains the maximum rate at which the Kafka producer can produce the data to a Kafka topic while reading the records from the filesystem. The consumer lag for each topic in the pipeline is monitored over a specific period. The following plot (Fig 5.6) shows the lag observed in each topic. Notably, the topic that the LEWS-GEOLOCATION-IDENTIFIER step reads from indicates an increasing consumer lag. This finding implies that this particular stage in the pipeline is the bottleneck, hindering the overall performance of the pipeline. The efficiency and throughput of the pipeline can be improved by locating and eliminating these bottlenecks.

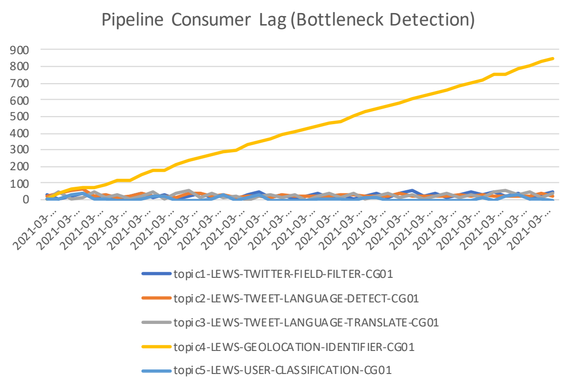


Fig. 5.6 Pipeline Consumer Lag

Subsequent experiments were conducted by removing the step LEWS-GEOLOCATION-IDENTIFIER from the pipeline, and it shows (Fig , there is no bottleneck in the pipeline which hinders its throughput.

The bottleneck in the process was identified, and subsequent experiments were run without the LEWS-GEOLOCATION-IDENTIFIER. These tests aimed to measure how much this process stage affected the pipeline's efficiency as a whole. The pipeline's throughput increased after removing the bottleneck shown in Fig 5.7). Increasing the parallelism of the LEWS-GEOLOCATION-IDENTIFIER step is a way to boost the pipeline's performance. As a result, the system could process more data and reduce wait times.

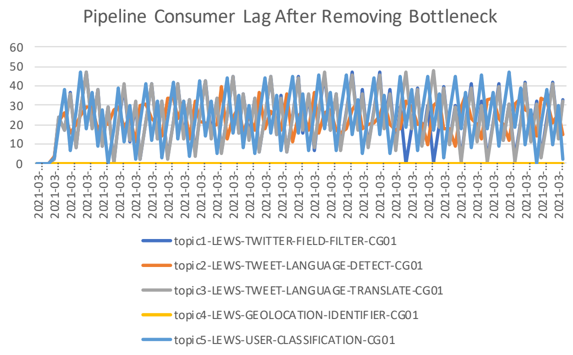


Fig. 5.7 Pipeline Consumer Lag after removing bottleneck

Another experiment with three different data rates was conducted to examine the sensitivity of consumer lag to data rates. The data rates were progressively increased in two stages to gain a more comprehensive understanding of the pipeline's response to varying data rates. This experimental setup was designed to elucidate the correlation between consumer lag and data rates, which could reveal helpful information about the pipeline's efficacy and scalability. This also helps identify which steps need to be scaled more when data rate is increased.

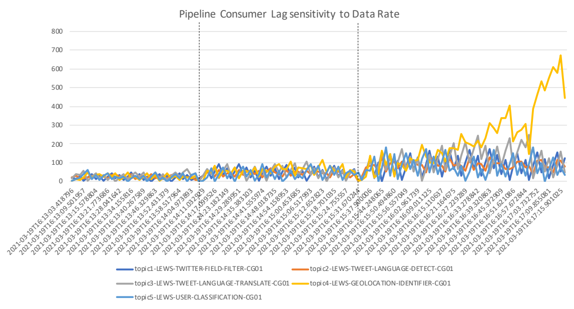


Fig. 5.8 Consumer Lag sensitivity to Datarate

5.6 Discussion

Given the research context, the investigation was aimed at designing and testing a distributed real-time analytics pipeline for early warning systems and Smart City projects. This work's contributions include evaluating a Landslip Early Warning System using social media data and real-world sensor data in real-time and designing a system to orchestrate streaming workflows distributed across multiple processing units connected using a common streaming solution. Early warning systems and smart city IoT applications are some of the real-world use cases that can benefit from this stream data pipeline system. In addition, utilising human sensors and contextual information from social media feeds, the Internet of Everything's (IoE) integration of people and processes could improve the precision of prediction and decision-making.

The proposed pipeline system addresses the near real-time data processing requirement from different IoE sources at variable data rates. The system can distribute the processing load by using Apache Kafka as a data backbone for streaming and storing data from different sources. The system's ability to perform complex processing on the stream and to pipeline the data processing tasks across multiple platforms and machines is made possible by the actor-based processing approach. Bottlenecks like the one discovered in the LEWS-GEOLOCATION-IDENTIFIER step can be efficiently monitored and removed to scale the proposed pipeline system. The pipeline can adjust to different workloads and maintain a high level of performance by monitoring consumer lag and adjusting the parallelism of processing steps. This flexibility is very important in Early Warning Systems and Smart City applications, which depend on prompt reactions to events and the effective management of large volumes of data generated by IoT devices. However, in this work, no automatic bottleneck mitigation techniques were proposed.

Nonetheless, the system and the experiments in this work have some limitations:

- Only one data source, Twitter, was considered to keep the experimentation simple. The evaluation was not thorough enough to judge the pipeline's effectiveness when subjected to a wide variety of complex data streams, despite the fact that it was intended to accommodate multiple data sources.

- Since only three machines were used in the experiments, the results may not be indicative of the system's scalability and performance in large-scale, real-world deployments.

- It was required to manually collect many tweets for the experiments as the Twitter streaming API only provides a fixed number of tweets per minute. This method might not represent the real data rate and volume the pipeline would need to process.

- These experiments did not objectively measure the pipeline's orchestration's effectiveness, especially in resource management and the distribution of processing tasks. The effectiveness of the proposed system's orchestration is thus still uncertain.

Future research that pays particular attention to these questions will shed light on the pipeline system's orchestration capabilities, paving the way for creating more reliable, scalable, and effective pipelines for streaming data to be used in things like early warning systems and smart city IoT applications.

Chapter 6

Conclusion and Future Work

This thesis discussed several methodologies for data flow management in Internet of Things(IoT) systems used in Smart Cities and Early Warning System (EWS), which consists of edge and cloud layers. It also discusses the distributed real-time analytics pipeline orchestration for systems like EWS and Smart Cities. The goal was to address the challenges of processing massive amounts of data generated by IoT systems while ensuring data freshness and efficiently coordinating the data flow from many heterogeneous devices.

Revisiting the original research questions presented at the beginning of the thesis, the following contributions have been made through this work.

1. What are the methodologies for achieving the maximum data flow rate for each node in the IoT network utilizing the available bandwidth while maintaining the stability of the network and the nodes?

   Chapter 3 discusses an IoT Emulation toolkit which enabled experimentation on three layered IoT networks. The proposed system and PoC is an emulation framework which closely models the real-world network and IoT devices.

   In Chapter 4, through the development of a formal model and a set of algorithms for managing data flow in IoT devices (Section 4.3), this thesis demonstrated how to allocate data rates based on the priority of IoT sensors/devices and adaptively adjust data flow rates based on available bandwidth (Section 4.3.2). In addition, an IoT Emulation Toolkit (Section 3.4) was developed to emulate a three-layer IoT network to conduct experiments and evaluate the performance of IoT networks. This toolkit assists in identifying backpressure, managing data flow, and maintaining data freshness, thereby ensuring network and node stability.

2. How to identify data flow bottlenecks in a three-layered IoT network dynamically?

   This thesis identified performance limitations and bottlenecks in IoT networks, such as backpressure, through developing and applying the IoT Emulation Toolkit and the backpressure detection method (Section 3.7.2). These experiments led to the development of data flow management algorithms (Section 4.3).

3. What are the methodologies for efficiently coordinating a large number of heterogeneous edge nodes and IoT devices sending data to the cloud simultaneously?

   The adaptive flow control algorithms presented in this thesis (Section 4.3) enable efficient coordination of heterogeneous edge nodes and IoT devices simultaneously sending data to the cloud by dynamically managing data flow rates based on available bandwidth and node priorities. This strategy helps preserve network stability and ensures efficient data flow management in IoT networks.

4. How can a distributed analytics pipeline help in Early Warning Systems like landslide early warning systems, and how can the pipelines be efficiently orchestrated?

   Chapter 5 of the dissertation presented a method for orchestrating a Big Data analytics processing pipeline for social media data processing in Early Warning Systems and Smart Cities. This methodology involved the design of a pipeline for streaming processing that addresses the orchestration challenges of a real-time streaming analytics pipeline. The proposed pipeline system satisfies the need for near-real-time data processing from multiple IoE sources with variable data rates. The system can distribute the processing load using Apache Kafka as the data backbone for streaming from various sources. The actor-based processing approach enables the system to perform complex processing on the stream and to pipeline the data processing tasks across multiple platforms and machines. This enables the distribution and orchestration of the analytics pipeline in a decoupled way based on the processing requirements.

This thesis provided valuable insights and solutions for managing data flow in IoT networks, identifying and addressing bottlenecks in three-layered IoT networks, coordinating large numbers of heterogeneous edge nodes and IoT devices, and designing and orchestrating distributed analytics pipelines for Early Warning Systems. The findings and contributions presented in this thesis and the IoT Emulation Toolkit can be a foundation for future research and development in IoT networks and analytics for EWS and Smart Cities.

Even though this work provides valuable methods for managing data flow in IoT networks and orchestrating distributed analytics pipelines, the following limitations must be acknowledged.

1. The IoT Emulation Toolkit allows for emulating IoT devices and edge nodes, but it may not precisely reflect the properties and limitations of real-world things and network conditions. Emulation toolkit experiment results may not accurately represent the performance or difficulties encountered by actual IoT devices.

2. Experiments focused on backpressure detection, data flow rate, and data freshness, and the toolkit was developed to simulate a three-layer IoT network. The experiments did not consider advanced QoS parameters. Even though the Linux TC tool is used to induce bandwidth limitations and latency, experiments used LAN networking, which allowed for relatively stable network bandwidth, latency, and jitter. The system's performance and data flow management may be affected by the more dynamic and varying network conditions likely to be present in real-world IoT networks.

3. Early Warning Systems can benefit significantly from integrating social media data into their streaming analytics pipeline. A deeper understanding of the pipeline's efficacy in real-world applications can be attained through a more exhaustive evaluation considering a wide range of data sources and complex data streams.

4. Data streaming pipeline bottlenecks are identified, but no automatic mitigation strategies for improving the system's scalability or performance are proposed. Future work will investigate the automatic mitigation of bottlenecks in distributed real-time stream processing pipelines.

5. Experiments in Chapters 4 and 5 that required manual data collection may not have represented the data rate and volume the pipeline would need to process in practical settings.

6.1 Future Work

The IoT emulation tool kit enabled us to perform experiments on the hardware directly emulating data flow from a real sensor network. Several enhancements to the tool kit are proposed to help the user to conduct IoT experiments efficiently. The IoT Emulation Toolkit developed in this work has room for improvement, particularly in adding features and making it more realistic. The adaptability of the toolkit would increase, for instance, if it supported multiple types of Internet of Things devices with varying capacities and communication standards than just MQTT used in this case. The ability to model more complex network behaviour, including latency, jitter, and packet loss variations, would also provide a more accurate representation of real-world IoT networks and expand the scope of experimentation.

The prototype system we developed enabled the edge devices to control the data rate it forwards to the cloud environment by dynamically reconfiguring the IoT devices. The AFC algorithm is effective for recovering the system from the backpressure in the data flow. This approach helps to improve the data freshness in a congested IoT network. Future work includes the integration of sampling algorithms into EGK to perform sampling of the incoming data keeping the data quality high. EGK also opens the possibility of load balancing between multiple edge devices to reduce the load while maintaining the data quality. The adaptive flow control algorithms discussed in this thesis have room for improvement and expansion through machine learning and reinforcement learning. These methods enable more advanced algorithms to detect changes in the data flow, anticipate potential bottlenecks and adapt the data or sampling rate in real time.

Creating adaptive data control algorithms that can distribute work between edge devices is another promising area for future study. Network performance and reliability can be significantly improved by intelligently distributing the data flow among multiple edge devices. To ensure effective processing and reduce the impact of a failure or congestion on one of the edge devices, the data flow could be dynamically rerouted through the nearest available edge device. New load-balancing strategies that consider IoT networks' specific features and limitations, like limited bandwidth and low power consumption requirements, could be the focus of future studies in this area. Data priorities and real-time network conditions are a few considerations that could be incorporated into such strategies, along with edge devices' processing power, energy consumption, and communication latency.

Further optimisation of the real-time analytics pipeline described in Chapter 5 could increase its performance and scalability. The pipeline's adaptability to massive data volumes and varying data rates could be improved by creating novel data partitioning, load balancing, and resource allocation strategies. This work finds bottlenecks in the data streaming pipeline steps but does not suggest any automatic mitigation strategies to improve the scalability and performance of the system. The future direction of this work includes the development of automatic backpressure detection and mitigation algorithms for real-time streaming pipelines.

References

[Apa] Apache nifi. Accessed on: 2023-05-07.

[che] Chef and puppet | chef.

[pup] Puppet infrastructure & it automation at scale.

[spa] spacy - industrial-strength natural language processing in python. https://spacy.io/.

[5] Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. (2003). Aurora: a new model and architecture for data stream management. *the VLDB Journal*, 12(2):120–139.

[6] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., and Chen, Z. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/. Software available from tensorflow.org.

[7] Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., and Stoica, I. (2013). BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM.

[8] Agha, G. A. (1986). A model of concurrent computation in distributed systems. *the MIT Press*.

[9] Ahlers, D., Wienhofen, L. W., Petersen, S. A., and Anvaari, M. (2019). A smart city ecosystem enabling open innovation. In *Innovations for Community Services: 19th International Conference, I4CS 2019, Wolfsburg, Germany, June 24-26, 2019, Proceedings 19*, pages 109–122. Springer.

[10] Ahmed, A. A. and Eze, T. (2019). An actor-based runtime environment for heterogeneous distributed computing. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 37–43. The Steering Committee of The World Congress in Computer Science, Computer âĂę.

[11] Akidau, T., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D., Nordstrom, P., and Whittle, S. (2013). Millwheel: fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):1033–1044.

[12] Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., et al. (2015). The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803.

[13] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376.

[14] Albino, V., Berardi, U., and Dangelico, R. M. (2013). Smart cities: definitions, dimensions, and performance.

[15] Almesberger, W. et al. (1999). Linux network traffic controlâĂŤimplementation overview.

[Apache] Apache. Apache/oozie: Mirror of apache oozie. Apache Software Foundation.

[17] Babar, M., Khan, M. S., Ali, F., Imran, M., and Shoaib, M. (2021). Cloudlet computing: recent advances, taxonomy, and challenges. *IEEE Access*, 9:29609–29622.

[18] Bagula, B. and Erasmus, Z. (2015). Iot emulation with cooja. In *ICTP-IoT workshop*, page 99.

[19] Bahl, L. R., Brown, P. F., de Souza, P. V., and Mercer, R. L. (1989). A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):1001–1008.

[20] Bahreini, T. and Grosu, D. (2017). Efficient placement of multi-component applications in edge computing systems. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 5. ACM.

[21] Barga, R. S., Goldstein, J., Ali, M., and Hong, M. (2006). Consistent streaming through time: A vision for event stream processing. *arXiv preprint cs/0612115*.

[22] Basher, R. (2006). Global early warning systems for natural hazards: systematic and people-centred. *Philosophical transactions of the royal society a: mathematical, physical and engineering sciences*, 364(1845):2167–2182.

[23] Beck, M., Bhatotia, P., Chen, R., Fetzer, C., Strufe, T., et al. (2017). Privapprox: privacy-preserving stream analytics. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pages 659–672.

[24] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[25] Bergenti, F., Poggi, A., and Tomaiuolo, M. (2014). An actor based software framework for scalable applications. In *Internet and Distributed Computing Systems: 7th International Conference, IDCS 2014, Calabria, Italy, September 22-24, 2014. Proceedings 7*, pages 26–35. Springer.

[26] Białecki, A., Muir, R., Ingersoll, G., and Imagination, L. (2012). Apache lucene 4. In *SIGIR 2012 workshop on open source information retrieval*, page 17.

[27] Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013a). Opentosca–a runtime for tosca-based cloud applications. In *Service-Oriented Computing: 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings 11*, pages 692–695. Springer.

[28] Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2013b). Tosca: portable automated deployment and management of cloud applications. In *Advanced Web Services*, pages 527–549. Springer.

[Boeing] Boeing. Boeing 787s to create half a terabyte of data per flight, says virgin atlantic. https://www.computerworlduk.com/data/boeing-787s-create-half-terabyte-of-data-per-flight-says-virgin-atlantic-3433595/. Accessed: 2019-04-08.

[30] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA. ACM.

[31] Borralho, R., Mohamed, A., Quddus, A. U., Vieira, P., and Tafazolli, R. (2021). A survey on coverage enhancement in cellular networks: Challenges and solutions for future deployments. *IEEE Communications Surveys & Tutorials*, 23(2):1302–1341.

[32] Borras, J. (2004). International technical standards for e-government. *Electronic journal of e-government*, 2(2):pp75–80.

[33] Bradley, J., Reberger, C., Dixit, A., Gupta, V., and Macaulay, J. (2013). Internet of everything (ioe): top 10 insights from ciscoâĂŹs ioe value at stake analysis for the public sector. *Economic Analysis*, pages 1–5.

[34] Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

[35] Buchmann, A. and Koldehofe, B. (2009). Complex event processing.

[36] Buyya, R., Ranjan, R., and Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *2009 international conference on high performance computing & simulation*, pages 1–11. IEEE.

[37] Byers, C. and Swanson, R. (2017). Openfog consortium openfog reference architecture for fog computing. *OpenFog Consortium Archit. Working Group, Fremont, CA, USA, Tech. Rep. OPFRA001*, 20817:27–28.

[38] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., and Tzoumas, K. (2015). Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering*, 38(4).

[39] Cavnar, W. B., Trenkle, J. M., et al. (1994). N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175. Citeseer.

[40] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., et al. (2003). Telegraphcq: Continuous dataflow processing for an uncertain world. In *Cidr*, volume 2, page 4.

[41] Chang, H., Hari, A., Mukherjee, S., and Lakshman, T. (2014). Bringing the cloud to the edge. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 346–351. IEEE.

[42] Chatfield, A. and Brajawidagda, U. (2012). Twitter tsunami early warning network: a social network analysis of twitter information flows.

[43] Cheng, B., Longo, S., Cirillo, F., Bauer, M., and Kovacs, E. (2015). Building a big data platform for smart cities: Experience and lessons from santander. In *2015 IEEE International Congress on Big Data*, pages 592–599. IEEE.

[44] Chippa, V. K., Chakradhar, S. T., Roy, K., and Raghunathan, A. (2013). Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, page 113. ACM.

[45] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

[Common Crawl] Common Crawl. Common Crawl. http://commoncrawl.org/.

[47] Cugola, G. and Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):1–62.

[48] Cunningham, H., Tablan, V., Roberts, A., and Bontcheva, K. (2013). Getting more out of biomedical documents with gate's full lifecycle open source text analytics. *PLoS computational biology*, 9(2):e1002854.

[49] De, S., Zhou, Y., Larizgoitia Abad, I., and Moessner, K. (2017). CyberâĂŞphysicalâĂŞ-social frameworks for urban big data systems: A survey. *Applied Sciences*, 7(10).

[50] Etzion, O. and Niblett, P. (2010). *Event processing in action*. Manning Publications Co.

[51] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114âĂŞ131.

[52] Evans, D. (2011). The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11.

[53] Frampton, M. and Frampton, M. (2018). Apache mesos. *Complete Guide to Open Source Big Data Stack*, pages 97–137.

[54] Friedewald, M. and Raabe, O. (2011). Ubiquitous computing: An overview of technology impacts. *Telematics and Informatics*, 28(2):55–65.

[55] Garcia, C. and Fearnley, C. J. (2012). Evaluating critical links in early warning systems for natural hazards. *Environmental Hazards*, 11(2):123–137.

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

[57] Goiri, I., Bianchini, R., Nagarakatte, S., and Nguyen, T. D. (2015). Approxhadoop: Bringing approximations to mapreduce frameworks. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 383–397. ACM.

[58] Golab, L. and Özsu, M. T. (2003). Issues in data stream management. *ACM Sigmod Record*, 32(2):5–14.

[59] Gormley, C. and Tong, Z. (2015). *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. " O'Reilly Media, Inc.".

[60] Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., and Buyya, R. (2017). ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296.

[61] Hesse, G. and Lorenz, M. (2015). Conceptual survey on data stream processing systems. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 797–802. IEEE.

[62] Hewitt, C., Bishop, P., and Steiger, R. (1973). Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence. In *Advance Papers of the Conference*, volume 3, page 235. Stanford Research Institute Menlo Park, CA.

[63] Honarvar, A. R. and Sami, A. (2019). Towards sustainable smart city by particulate matter prediction using urban big data, excluding expensive air pollution infrastructures. *Big data research*, 17:56–65.

[64] Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2006). Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60.

[65] Hunkeler, U., Truong, H. L., and Stanford-Clark, A. (2008). MQTT-S - A publish/subscribe protocol for wireless sensor networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*, pages 791–798. IEEE.

[66] Intrieri, E., Gigli, G., Mugnai, F., Fanti, R., and Casagli, N. (2012). Design and implementation of a landslide early warning system. *Engineering Geology*, 147-148:124–136.

[67] James, P. M., Dawson, R. J., Harris, N., and Joncyzk, J. (2014). Urban observatory environment. *Newcastle University*, pages 154300–19.

[68] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

[69] Kandula, S., Shanbhag, A., Vitorovic, A., Olma, M., Grandl, R., Chaudhuri, S., and Ding, B. (2016). Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *Proceedings of the 2016 international conference on management of data*, pages 631–646. ACM.

[70] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

[71] Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[72] Kreps, J., Narkhede, N., Rao, J., et al. (2011). Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, volume 11, pages 1–7.

[73] Krishnamurthi, R., Kumar, A., Gopinathan, D., Nayyar, A., and Qureshi, B. (2020). An overview of iot sensor data processing, fusion, and analysis techniques. *Sensors*, 20(21):6076.

[74] Kryvasheyeu, Y., Chen, H., Obradovich, N., Moro, E., Van Hentenryck, P., Fowler, J., and Cebrian, M. (2016). Rapid assessment of disaster damage using social media activity. *Science advances*, 2(3):e1500779.

[75] Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*.

[76] Light, R. A. et al. (2017). Mosquitto: server and client implementation of the mqtt protocol. *J. Open Source Software*, 2(13):265.

[77] Lima, L. E., Kimura, B. Y. L., and Rosset, V. (2019). Experimental environments for the internet of things: A review. *IEEE Sensors Journal*, 19(9):3203–3211.

[78] Lin, W., Qian, Z., Xu, J., Yang, S., Zhou, J., and Zhou, L. (2016). Streamscope: continuous reliable distributed processing of big data streams. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 439–453.

[79] Locke, D. (2010). Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library*, 15.

[80] Looga, V., Ou, Z., Deng, Y., and Ylä-Jääski, A. (2012a). Mammoth: A massive-scale emulation platform for internet of things. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, volume 3, pages 1235–1239. IEEE.

[81] Looga, V., Ou, Z., Deng, Y., and YlÃď-JÃďÃďski, A. (2012b). Mammoth: A massive-scale emulation platform for internet of things. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, volume 03, pages 1235–1239.

[82] Loper, E. and Bird, S. (2002). Nltk: the natural language toolkit. *arXiv preprint cs/0205028*.

[83] Lucene, A. (2010). Apache lucene-overview. *Internet: http://lucene. apache. org/iava/docs/[Jan. 15, 2009]*.

[84] Luckham, D. C. (2002). *The power of events: An introduction to complex event processing in distributed enterprise systems*. Addison-Wesley Longman Publishing Co., Inc.

[85] Lukić, M., Mihajlović, Ž., and Mezei, I. (2018). Data flow in low-power wide-area iot applications. In *2018 26th Telecommunications Forum (TELFOR)*, pages 1–4. IEEE.

[86] Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 276–283. Association for Computational Linguistics.

[87] Mannila, H. and Räihä, K.-J. (1992). *The design of relational databases*. Addison-Wesley Longman Publishing Co., Inc.

[88] Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

[89] Marjani, M., Nasaruddin, F., Gani, A., Karim, A., Hashem, I. A. T., Siddiqa, A., and Yaqoob, I. (2017). Big iot data analytics: architecture, opportunities, and open research challenges. *ieee access*, 5:5247–5261.

[90] Markit, I. (2017). News release | ihs markit online newsroom.

[91] McLoughlin, S., Maccani, G., Puvvala, A., and Donnellan, B. (2021). An urban data business model framework for identifying value capture in the smart city: The case of organicity. *Smart Cities and Smart Governance: Towards the 22nd Century Sustainable City*, pages 189–215.

[92] Meyerson, J. (2014). The go programming language. *IEEE Software*, 31(5):104–104.

[93] Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

[94] Miraz, M. H., Ali, M., Excell, P. S., and Picking, R. (2015). A review on internet of things (iot), internet of everything (ioe) and internet of nano things (iont). In *2015 Internet Technologies and Applications (ITA)*, pages 219–224.

[95] Mohammadi, M., Al-Fuqaha, A., Sorour, S., and Guizani, M. (2018). Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960.

[96] Morabito, R. and Beijar, N. (2016). Enabling data processing at the network edge through lightweight virtualization technologies. In *2016 IEEE International Conference on Sensing, Communication and Networking (SECON Workshops)*, pages 1–6.

[97] Morrison, J. P. (2010). *Flow-Based Programming: A new approach to application development*. CreateSpace.

[98] Morshed, A., Jayaraman, P. P., Sellis, T., Georgakopoulos, D., Villari, M., and Ranjan, R. (2017). Deep osmosis: Holistic distributed deep learning in osmotic computing. *IEEE Cloud Computing*, 4(6):22–32.

[99] Mukherjee, A., Rojas, B., and Ujhazy, H. (2020). Iot growth demands rethink of long-term storage strategies, says idc.

[100] Murray, D. G., McSherry, F., Isaacs, R., Isard, M., Barham, P., and Abadi, M. (2013). Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 439–455. ACM.

[101] Nam, T. and Pardo, T. A. (2011). Conceptualizing smart city with dimensions of technology, people, and institutions. In *Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times*, pages 282–291.

[102] Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., and Campbell, R. H. (2017). Samza: stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment*, 10(12):1634–1645.

[103] O'Keeffe, D., Salonidis, T., and Pietzuch, P. (2018). Frontier: resilient edge processing for the internet of things. *Proceedings of the VLDB Endowment*, 11(10):1178–1191.

[104] Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006). Cross-level sensor network simulation with cooja. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648.

[105] Patel, N., Mehtre, B., and Wankar, R. (2019a). Simulators, emulators, and test-beds for internet of things: A comparison. In *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 139–145. IEEE.

[106] Patel, N. D., Mehtre, B. M., and Wankar, R. (2019b). Simulators, emulators, and test-beds for internet of things: A comparison. In *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 139–145.

[107] Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[108] Perwej, Y., Kerim, B., Adrees, M. S., and Sheta, O. E. (2017). An empirical exploration of the yarn in big data. *International Journal of Applied Information Systems (IJAIS)*, 12(9):19–29.

[109] Phengsuwan, J., Thekkummal, N. B., Shah, T., James, P., Thakker, D., Sun, R., Pullarkatt, D., Hemalatha, T., Ramesh, M. V., and Ranjan, R. (2019). Context-based knowledge discovery and querying for social media data. In *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 307–314. IEEE.

[110] Qasha, R., Cala, J., and Watson, P. (2015). Towards automated workflow deployment in the cloud using tosca. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 1037–1040. IEEE.

[111] Qian, B., Su, J., Wen, Z., et al. (2020). Orchestrating the development lifecycle of machine learning-based iot applications: A taxonomy and survey. *ACM Computing Surveys (CSUR)*, 53(4):1–47.

[112] Quoc, D. L., Chen, R., Bhatotia, P., Fetze, C., Hilt, V., and Strufe, T. (2017). Approximate stream analytics in Apache Flink and Apache Spark Streaming. *arXiv preprint arXiv:1709.02946*.

[113] Ranjan, R., Garg, S., Khoskbar, A. R., Solaiman, E., James, P., and Georgakopoulos, D. (2017). Orchestrating bigdata analysis workflows. *IEEE Cloud Computing*, 4(3):20–28.

[114] Ranjan, R., Rana, O., Nepal, S., Yousif, M., James, P., Wen, Z., Barr, S., Watson, P., Jayaraman, P. P., Georgakopoulos, D., et al. (2018). The next grand challenges: Integrating the internet of things and data science. *IEEE Cloud Computing*, 5(3):12–26.

[115] Rossi, C., Acerbo, F. S., Ylinen, K., Juga, I., Nurmi, P., Bosca, A., Tarasconi, F., Cristoforetti, M., and Alikadic, A. (2018). Early detection and information extraction for weather-induced floods using social media streams. *International journal of disaster risk reduction*, 30:145–157.

[116] Sajjad, H. P., Danniswara, K., Al-Shishtawy, A., and Vlassov, V. (2016). Spanedge: Towards unifying stream processing over central and near-the-edge data centers. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 168–178. IEEE.

[117] Shelby, Z., Hartke, K., and Bormann, C. (2014). The constrained application protocol (coap). Technical report.

[118] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee.

[119] Silva, B. N., Khan, M., and Han, K. (2018). Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities. *Sustainable cities and society*, 38:697–713.

[120] Sonmez, C., Ozgovde, A., and Ersoy, C. (2018). Edgecloudsim: An environment for performance evaluation of edge computing systems. *Transactions on Emerging Telecommunications Technologies*, 29(11):e3493.

[121] Stals, S., Smyth, M., and Mival, O. (2019). Urbanixd: From ethnography to speculative design fictions for the hybrid city. In *Proceedings of the Halfway to the Future Symposium 2019*, pages 1–10.

[122] Standard, O. (2012). Oasis advanced message queuing protocol (amqp) version 1.0. *International Journal of Aerospace Engineering Hindawi www. hindawi. com*, 2018.

[123] Standard, O. (2014). Mqtt version 3.1. 1. *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, 1:29.

[124] Standard, O. (2019). Mqtt version 5.0. *Retrieved June*, 22:2020.

[125] Stanford-Clark, A. and Truong, H. L. (2013). Mqtt for sensor networks (mqtt-sn) protocol specification. *International business machines (IBM) Corporation version*, 1(2):1–28.

[126] Stoller, M. H. R. R. L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., and Lepreau, J. (2008). Large-scale virtualization in the emulab network testbed. In *USENIX annual technical conference, Boston, MA*, pages 255–270.

[127] Sundani, H., Li, H., Devabhaktuni, V., Alam, M., and Bhattacharya, P. (2011). Wireless sensor network simulators a survey and comparisons. *International Journal of Computer Networks*, 2(5):249–265.

[128] Sundmaeker, H., Guillemin, P., Friess, P., and Woelfflé, S. (2010). Vision and challenges for realising the internet of things. *Cluster of European research projects on the internet of things, European Commision*, 3(3):34–36.

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

[130] Tönjes, R., Barnaghi, P., Ali, M., Mileo, A., Hauswirth, M., Ganz, F., Ganea, S., Kjærgaard, B., Kuemper, D., Nechifor, S., et al. (2014). Real time iot stream processing and large-scale data analytics for smart city applications. In *poster session, European Conference on Networks and Communications*, page 10. sn.

[131] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al. (2014). Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM.

[132] Union, I. T. (2012). Internet of things global standards initiative.

[133] UNISDR, P. (2006). Global survey of early warning systems: An assessment of capacities, gaps and opportunities toward building a comprehensive global early warning system for all natural hazards. *Platf Promot Early Warn UNISDRâĂŤPPEW UN*, page 2006.

[134] Upton, E. and Halfacree, G. (2014). *Raspberry Pi user guide*. John Wiley & Sons.

[135] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. (2013). Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–16.

[136] Venkataraman, S., Panda, A., Ousterhout, K., Armbrust, M., Ghodsi, A., Franklin, M. J., Recht, B., and Stoica, I. (2017). Drizzle: Fast and adaptable stream processing at scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 374–389. ACM.

[137] Villari, M., Fazio, M., Dustdar, S., Rana, O., and Ranjan, R. (2016). Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83.

[138] Weiser, M. (1991). The computer for the 21 st century. *Scientific american*, 265(3):94–105.

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

[140] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*, pages 423–438. ACM.

[141] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

[142] Zeng, X., Garg, S. K., Strazdins, P., Jayaraman, P. P., Georgakopoulos, D., and Ranjan, R. (2017). Iotsim: A simulator for analysing iot applications. *Journal of Systems Architecture*, 72:93–107.

Appendix A

Additional Images

A.1 Landslip EWS Data Processing

A.1.1 In-Stream Natural Language Processing

Natural Language Processing (NLP) is a set of information engineering techniques which enables computers to process and make sense of human (natural) languages. NLP technique has evolved from complex handwritten rules to models trained using machine learning. Earlier machine learning techniques like decision trees[19, 86] generated rules similar to handwritten ones, using machine learning. The application of NLP in this work is to extract useful information from the natural language and to classify the content into different topics of interest. Language modelling techniques apply probability distribution over a sequence of words. Unigram, n-gram [34, 39], Exponential and Neural networks [93, 24, 71] are the main types of language models in use. Recent studies promise high accuracy in classifying natural language using a neural network. A unified architecture for NLP using deep learning technique has been introduced in work [45] by NEC Labs. In this work, the input sentence can be processed to perform part-of-speech tagging, chunking, named entity tags, semantic roles etc. using a language model and CNN. A study [70] at New York University reveals a series of experiments using Convolutional Neural Network (CNN) which is trained on a pre-trained model of word vectors for sentence classification in which the model showed significant improvement in performance in several NLP tasks. Over the years several open-source NLP projects like NLTK[82], CoreNLP[88], Spacy[spa], GATE[48] etc. gained interest of both academia and industry. While these methods and tools support natural language processing, building knowledge from natural language pose several challenges.

1. Natural Language Processing Pipeline
2. Real-time Information Extraction
    (a) Named Entity Recognition
    (b) Geo-Location Identification

A.1.2 Social Media Content Classification

The first step to process the user text is to identify the hazard which the user is referring to on social media. Recurrent Neural Network (RNN) [75] such as Long Short Term Memory networks (LSTM) and Convolutional Neural Network (CNN) are widely used in text classification. In this work, we have used CNN, a deep learning technique, [70] to perform text classification. A model, which is custom trained using weather related text is used. The model training and inference processes are explained in the following sections.

Information Extraction and Annotation

In this step, a scenario is developed using information about the situation from user-provided text. NLP techniques, namely Part of Speech (PoS) tagging and Named Entity Recognition (NER) are used to extract useful information from the text. Pre-trained NLP models for the English language recognise Geo-location and affected entities (for example, Road, Building, Electric Pole, etc.). The English language model,a multi-task CNN trained on OntoNotes[64], with GloVe[107] vectors trained on Common Crawl[Common Crawl] is used in this work. This model is built for assigning word vectors, context-specific token vectors, POS tags, dependency parse and named entity recognition.

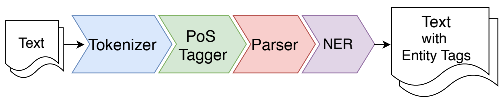


Fig. A.1 NLP Pipeline for Named Entity Recognition

Named entities are extracted from the user-generated content using NER. An NLP tool called Spacy [spa] was used to perform the series of tasks required to perform NER. The processing pipeline consists of a tokenizer, PoS tagger, Parser, and NER. The tokenizer tokenizes sentences into words for which a PoS tag is attached based on the sentence structure. Then the parser performs a dependency parsing of the sentence, which represents its grammatical structure and defines the relationship between words. This step is followed by the NER phase, which identifies the type of entity such as geo-location, person, organisation, physical object, date, time, building/infrastructure etc. This model and pipeline gave a largely accurate prediction of the type of the entity from the noun words tagged from the PoS tagging phase. Figure A.2 shows an example of a sentence being processed and labelled. The entity

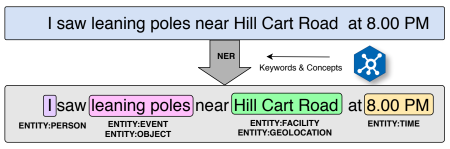


Fig. A.2 Named Entity Recognition Example

tags are attached to the original data as metadata for storage and indexing. This extracted information is instantiated as objects based on the concepts defined in the ontology and stored in the knowledge base.

Model

A classification hierarchy, as shown in Figure A.3, has been defined for our prototype system. A model is trained using 1000 user-generated texts related to hazards, which are marked for different hazard events and warning signs (for example, Events: Flood, Heavy Rainfall, Snow etc.; Warning Signs: Leaning Light Pole, Water Discolouration etc.). Each class has around 200 records. TensorFlow [6], an open source library, was used for data preparation, training and inference. The model is similar to the one proposed by Kim Yoon in his work Convolutional Neural Networks for Sentence Classification [70], which achieved good classification performance for different text classification tasks like sentiment analysis and is a standard baseline for new text classification methods. The model consists of a word embedding layer, which maps vocabulary word indices to lower dimensional vector spaces. The convolutional layer calculates convolutions over the embedded word vectors using different filter sizes as each convolution produces tensors of different shapes followed by max-pooling, which is a sampling-based discretization process. These vectors are later merged to form a large feature vector. Full details of the CNN layers and training process are beyond the scope of this paper.

Inference

Every text message received by the Landslip agent is passed to the classifier, which outputs the hazard event and any warning sign mentioned in the text. This step enables the system to understand the topic from the user-generated content. Data classification for this system is a two-step process involving the classification of hazards and warning signs. In the first step, the classifier tags the message whether a hazard or warning sign is present in the text. The second step involves two classifiers, one for classifying the type of hazard and the second for categorising the kind of warning sign as per the classification hierarchy. In some cases, a message may contain information about both hazard and warning sign. In such a scenario, the system passes this message to both classifiers. In the inference step, the result is attached as metadata to the input text.

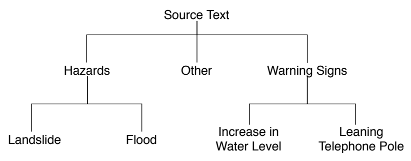


Fig. A.3 Data Classification Hierarchy

A.1.3 Mapping with Landslip Ontology

The Landslip Ontology is a conceptual model that formally represents domain knowledge about landslides captured from domain experts of natural hazards management. Th ontology consists of concepts and relationships but does not model concrete objects or *named individuals*, that represent actual events of landslides. With the emergence of social media as a potential resource to build the domain knowledge, social media contents to represent actual events of landslides are dynamically instantiated within the ontology. In order to do this however, sophisticated techniques are required to understand the context of the social media content and extract information from the content to create individuals based on the conceptual model. Figure A.4 shows the process o

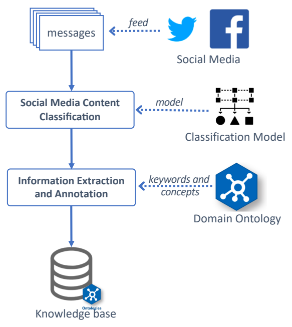


Fig. A.4 Process of populating the Knowledge Base from social media content