Middleware Techniques in A Rapid Response System in Wireless Sensor Networks

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by

Yuechun Wang

May 2021

Abstract

Wireless Sensor Networks (WSNs) are composed of embedded computers equipped with sensors, actuators and low-power radios that self-organise to form wireless networks capable of sensing the physical world. Despite their proven efficacy, modern WSNs uptake remains limited. This is primarily due to the applications' growing demand on the valid information extracted from the timeliness data flow and the complexity of data processing on the massive dynamic sensed data on motes. Two research problems could be addressed. The first one is how to ensure the effectiveness of the information extracted from the real-time sensed data; the second one is how to address the complex computing problem on the WSN nodes with limited computing ability.

In the distributed Internet architecture, edge computing technology has outstanding performances in solving the problem of rapid network service response. It provides computing, storage, and network bandwidth near data source or users. Meanwhile, as an extension of cloud computing, the application of fog computing proposed by Cisco in the WSN environment has extensively exerted the effectiveness of collaboration among the massive WSN nodes, thereby improving the computing efficiency of the node clusters. Suppose the advantages of edge computing can be brought into play in the WSN scenario and combined with the characteristics of fog computing. In that case, the core problem, which is how to balance the trade-off between the responding time of a WSN system and the computing complexity of data processing, can be solved.

Inspired by the research problems and potential solutions described above, the thesis explores several technologies that can take the characteristics of WSN and the requirements of logistics applications into account, such as hierarchical edge computing and Mobile Agent (MA). The hierarchical edge computing architecture proposed in the thesis in conjunction with the innovative rapid response strategy can ensure that under the premise of 80% coverage of real-time network nodes, data anomalies can be classified autonomously and responded within 20 sampling units, thereby reducing the delay caused by waiting for cloud computing results and decision communication. Besides, applying the mobile agent cooperation mechanism, a mobile agent-based middleware framework is presented in the thesis. Based on the randomly generated network topology, the middleware is tested under a variety of network scale with the node coverage by a MA applied patrol mechanism of above 98%.

Acknowledgements

First and foremost I am extremely grateful to my supervisors, Prof. Ka Lok Man, Prof. Steven Guan, Prof. Prudence Wong, and Prof. Danny Hughes for their invaluable advice, continuous support, and patience during my PhD study. Their immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life.I would also like to thank Dr. Kevin Lee and Dr. Jieming Ma for their technical support on my study.

I would like to thank all the members in SD428 and IR713. It is their kind help and support that have made my study and life in the XJTLU a wonderful time.

Finally, I would like to express my gratitude to my parents and my husband. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Objectives

The development of information technology with the Internet of Things (IoT) [16][17][30] as the core is known as the third information technology revolution. Since 2009, countries worldwide have been vigorously promoting the technology development and application of the Internet of Things.

Wireless Sensor Network (WSN) [16] is an infrastructure that can sense the changes of environmental data in the application scenarios of the Internet of Things and has significant research value and application prospect. In transport [61], agriculture [51][38], smart cities [44] and smart homes [23], and other intelligent application scenarios [46][5][40], WSN is playing a critical role. Tens of thousands of ubiquitous sensors enable WSNs to continuously capture large amounts of sensed data, which will keep on growing in the coming years. The WSN can continuously obtain massive real-time data from extensive coverage or even distributed application scenarios. Selective data retention, giving reasonable meaning to these data and making effective processing have become a hot research field, such as big data analytics.

However, in many practical WSN scenarios, such as the applications that ensure the safe storage of hazardous chemicals, the application is susceptible to data timeliness and the delay of data processing. This indicates that even though WSNs have a strong data capture ability to sense the environment with increasingly accuracy, give meaning to these enormous amounts of data, and how to use them intelligently and quickly in various mainstream applications are the challenges of current research [72]. The value of WSNs as the tentacles of IoT systems cannot be realised under inaccurate data analytics and delayed system response. Therefore, the real-time screening and efficient processing of these sensor data have become the hot research directions and challenges. Compared with the calculation and response time required by large-scale cloud computing and big data analysis, the real-time and fast, effective feedback that the system can make is more important.

In other words, it is a trade-off between computational complexity and response time of the WSN system. If the computing process is arranged in the cloud while the WSN nodes are only responsible for data collection, it requests longer response time to achieve more accurate calculation results. On the other hand, there are also many problems when using nodes alone for data processing. For example, due to nodes' limited computing power and having to consider the service life and battery replacement cost, data processing results cannot be as intelligent and reliable as cloud computing. Moreover, supposing all nodes in a massive WSN do the calculation and share the computing results with the one interested in, there will be severe channel congestion, information synchronisation and other communication problems.

The research challenges and questions mentioned above draw forth the two core objectives of the proposed research. The first one is a middleware technology supporting dynamic real-time WSN tasks by achieving the cooperation between sensor nodes. The second one is a framework supporting fast response system in a network architecture that is physically near the data source instead of on the cloud. In this dissertation, the warehouse environment in logistics scenario is selected as a typical rapid response application scenario. The mobile agent technology on the single node and the middleware platform supporting mobile agents operation are studied and discussed. Meanwhile, a hierarchical edge computing framework is proposed, which can be deployed in distributed warehouse application scenarios, and supports real-time processing of sensed data and rapid system response.

1.2 Contributions

To achieve the aforementioned core research objectives, the research presented in this thesis has contributed to several middleware techniques in a Rapid Response System in the WSN scenario. An overview of these contributions and their cooperation in the system is illustrated in the Figure 1.1.

The detailed points are presented as follows:

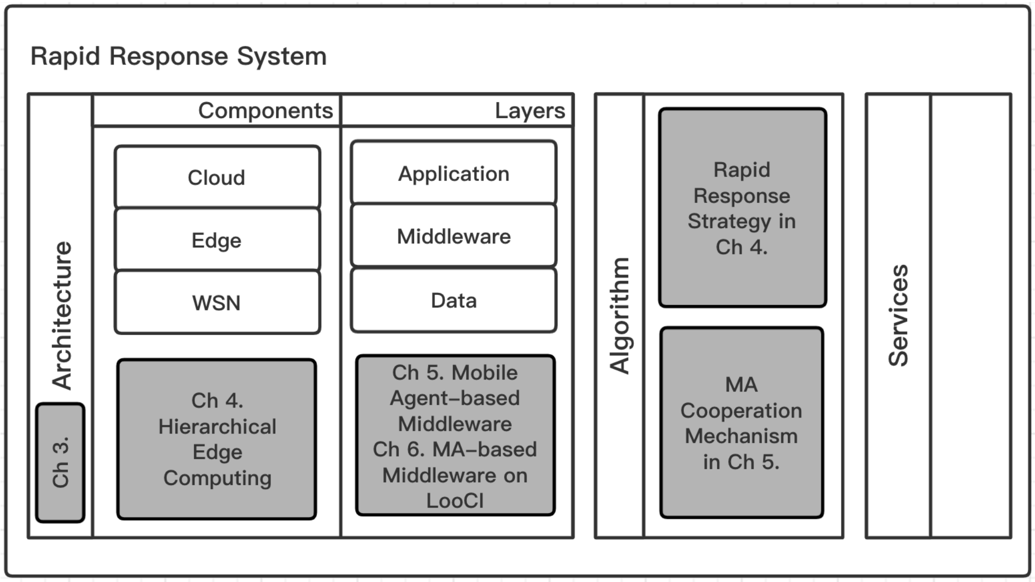


Figure 1.1: Overview of the contributions.

Hierarchical Edge Computing

1. A rapid response system architecture designed for outlier data detection is proposed in the thesis, which involves the concept of hierarchical edge computing. It brings the advantages of edge computing, its low latency, to the edge of the network. The performance of the system is evaluated by a software-defined simulation, with a concentration on the accuracy and rapidness of the system reaction on the edge nodes at different grades in the architecture.

2. An algorithm for distinguishing and rapidly responding to emergency cases on distributed warehouse management in logistics is proposed. By applying the proposed rapid response algorithm, above eighty per cent of the abnormal cases can be detected and responded to precisely, in less than twenty sampling units.

3. Three types of outlier data abstracts are designed and proposed to promote the portability of the system.

Mobile Agent-based Middleware

4. A mobile agent-based middleware framework designed for achieving the dynamic tasks by the cooperation between the sensor nodes in WSN is proposed.

5. A mobile agent cooperation mechanism for the general WSN environment is proposed based on the mobile agent-based middleware framework.

6. Being motivated by a practical hazardous chemical storage warehouse scenario, two universal mobile agents have proposed: the 'patrol agent' and the 'serve agent'.

7. Through the complementary functions of the MA and the suitable coordination mechanism, the general Internet of Things applications can be deployed and executed automatically. The applications achieved by coordinated mobile agents can respond to the environmental mutations in the first place and pre-process these mutations and anomalies without human intervention.

To sum up, the dissertation highlights contributions to several innovative WSN middleware techniques, such as hierarchical edge computing and mobile agent, supporting dynamic application tasks in a rapid response system. With these techniques and corresponding algorithms, the system can ensure that under the premise of 80% coverage of real-time network nodes, data anomalies can be classified autonomously and responded within short sampling units, thereby reducing the delay caused by waiting for centralised computing results and decision communication. With the mobile agents' node coverage of above 98%, the system is proved with the potential commercial value by testing on the randomly generated network topology and various network scales. Since the abnormal data types that used for the system's performance test is first proposed in the thesis, the repeating tests under the different sampling rates and the different network scales are done to make up for the leak of benchmark reference.

1.3 Publications

The thesis is partially based on the following publications:

Journal Paper

1. Y. Wang, K. L. Man, S. Guan, and D. Hughes, *A New Mobile Agent-based Middleware*

System Design for Wireless Sensor Network, IAENG Int. J. Comput. Sci., vol. 44, no. 4, pp. 591–597, 2017. [63]

This paper presents an innovative mobile agent-based middleware design, which could optimally achieve the intensive requirements of a WSN middleware. The middleware proposed in this paper has considered resources limitation issues that are commonly addressed on normal sensor nodes; meanwhile, it provides a possibility of platform independence and programming language independence. In addition, a mobile agent-based system can reduce network payload and dynamically adapt to environmental changes. Therefore, the middleware presented in this paper can perceive changes in the operating environment and automatically respond to these changes.

2. Y. Wang and K. L. Man, *A Mobile Agent Architecture for WSN Nodes with REST Advantages*, J. Ind. Inf. Technol. Appl. JIITA, vol. 2, no. 1, pp. 7–10, Mar. 2018, doi: 10.22664/ISITA.2018.2.1.7. [65]

IoT devices and nodes integrated with suitable mobile agents are deployed in various fields to collect data or monitor the systems. The rapid developing of software architecture highlights the usage of the distributed frameworks, which introduces Internet communication between nodes. The Constrained Application Protocol (CoAP) is a suitable Internet protocol applied to network-constrained systems. This paper illustrates how to enhance the architecture of such a framework based on CoAP to access and manage nodes with RESTful interfaces via the Internet.

3. Y. Wang, K. L. Man, K. Lee, D. Hughes, S.-U. Guan, and P. Wong, *Application of Wireless Sensor Network Based on Hierarchical Edge Computing Structure in Rapid Response System*, Electronics, vol. 9, no. 7, Art. no. 7, Jul. 2020, doi: 10.3390/electronics9071176. [67]

This paper presents a rapid response system architecture for the distributed management of warehouses in logistics by applying the concept of tiered edge computing. A tiered edge node architecture is proposed for the system to process computing tasks of different complexity, and a corresponding rapid response algorithm is introduced. The paper emphasises abstracted outlier sensing data classification, which could better match different sensing types and transplant to various application fields. A software-defined simulation evaluates the system performance on response time and response accuracy. It can be concluded that common predefined emergency cases can be detected and responded rapidly.

Conference Proceeding

1. Y. Wang, K. L. Man, S. Guan, and D. Hughes, *Feasibility Analysis of Achieving Mobile Agents for Wireless Sensor Network based on LooCI*, 2017, vol. 2228, pp. 714–718, [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85034436083&partnerID=40&md5=a4d792476a385c653382932022e454fb. [62]

2. Y. Wang and K. L. Man, *Mobile Agent Framework for WSN based on CoAP*, in Proceedings of the International Symposium on Innovation in Information Technology and Applications, Okinawa, Japan, Feb. 2019, vol. 1, p. 1. [64]

3. Y. Wang, K. L. Man, D. Hughes, S. Guan, and P. Wong, *A Rapid Response Approach Applying Edge Computing for Distributed Warehouses in WSN*, in Advanced Multimedia and Ubiquitous Engineering, Singapore, 2020, pp. 183–189, doi: 10.1007/978-981-32-9244-4_26.[66]

1.4 Thesis Outlines

The rest of the thesis is organised as follows:

Chapter 2 reviews the research background from four aspects, which are the value of Wireless Sensor Network in the domain of logistics; the existing middleware platforms that fit for WSN application scenarios and the emergent demand on middleware techniques that support for dynamic applications; the features of edge computing and the value of adopting edge computing to the WSN application scenarios; and the definition of mobile agent technique and its value on the dynamical applications.

Chapter 3 presents the overview of the proposed system architecture and network that applied the hierarchical edge computing framework and the mobile agent-based middleware on the single node, which has been used as the core system architecture throughout the entire research phase.

Chapter 4 presents the rapid response approach applying the hierarchical edge computing structure into the WSN application scenario. An application case study is presented to evaluate the rapid response ability of the system.

Chapter 5 presents the components of the proposed mobile agent and the supporting middleware platform on a single node. Based on the single node structure, this chapter also illustrates the cooperation mechanism of the mobile agents.

Chapter 6 presents the approach and tests on porting the mobile agent-based middleware depicted in Chapter 5 to an event-based component infrastructure - LooCI. Feasibility analysis and requirement analysis are presented as a proof-of-concept.

Chapter 7 concludes the research outcomes and the potential future works.

Chapter 2

Background

2.1 Wireless Sensor Networks in Logistics

As the tentacle to perceive the environment in the Internet of Things (IoT), Wireless Sensor Network (WSN) is a multi-hop self-organising network system composed of massive cheap micro sensor nodes deployed in the monitoring area. The sensor system aims to sense, collect and process the information from the sensing objects in the network by all the sensors cooperatively, and transfer the information to the observers and users.

The unique advantages of WSN are increasing in importance with the popularity of IoT concepts and applications. WSNs are applicable in small, medium, and large-scale network deployments. WSN is becoming a vital infrastructure for various applications, especially in the domains of natural disaster warning [14], industry [20], logistics [70][45], as well as the smart houses. WSNs have the ability to sense the environment and work autonomously - these features highlight both the research value and commercial value of WSN. However, the deployment cost and application mode of WSN determine their shortcomings that cannot be ignored. Node heterogeneity, limited storage, and limited power supply are common problems faced by WSN applications [39]. In addition, as the number of applications and development platforms increase, the low utilisation of resources becomes a new issue in the area [25].

Because of its characteristics such as no need to set up network facilities, rapid deployment and strong survivability, WSN is originated in the military field and has excellent military value [11]. Besides, WSN has promising applications in many fields, such as fine agriculture, intelligent transportation, biological medicine, environmental monitoring, emergency relief, etc. As a multi-faceted application scenario of WSN, logistics has many challenges and opportunities, such as equipment detection in production logistics, warehouse environment detection [42], tracking and detection of transport vehicles and materials in transit [69], dangerous goods logistics management and cold chain logistics management [35], etc.

Given the above logistics scenarios, the essential features of WSN include massive quantities of sensor nodes, node energy management, node positioning, and dynamic network management, which are also the core research value of WSN.

Because of the large number of nodes distributed in the sensor networks, wide coverage area as well as the complex working environment, it is not worthy or convenient to replace the battery for all the nodes [5]. In logistics, the monitoring of dangerous cargo is a typical case. In the design of sensor networks, energy-saving must be fully considered, so that nodes can survive for months or even years, and complete as many tasks as possible with less energy.

Besides, the choice of routing protocol for communication between nodes determines the reliability of data transmission and the level of communication energy consumption [36]. Choosing a routing protocol that saves energy can effectively prolong the lifetime of the network system. Based on this principle, the protocol should not be excessive complex to store massive state information and over-frequent exchange routing information between nodes. Simultaneously, redundant information should be avoided to reduce energy waste.

In addition, whether the sensor node is installed in the warehouse or on the transport vehicle, the node positioning is a problem that can not be ignored. Node localisation is to determine the relative or absolute position of sensor nodes. The data collected by the node must be combined with its position information in the coordinate measurement system to be meaningful for the applications [71]. For example, in logistics monitoring, if a monitoring value is found to be abnormal, then it is necessary to know which location can take corresponding measures to solve the problem.

Furthermore, Wireless Sensor Network used in logistics is a dynamic network. Some nodes can be moved, for instance the position of some nodes changes with vehicles' movement, which involves a mobile control problem. Mobile control is responsible for detecting and controlling the movement of nodes, maintaining routes to the convergence points [43], and enabling sensor nodes to track their neighbours [33]. A node may fail to connect the network due to exhaustion of battery power or other failures; it may also join the current network for specific application needs. All of these will influence the topology of the network, so network topology control is of particular importance.

2.2 Middleware

The meaning of the word “Dynamic” has two levels throughout the entire research process. The first level refers to the dynamic network formed by the movement of nodes in the spatial dimension. As mentioned in the previous section, the action of mobile nodes in logistics scenarios such as vehicles leads to the dynamic changes of the network, that is, the joining or quitting behaviour of nodes will change the topology of the network. The second level means that in the time dimension, the sensor data collected by the sensor nodes according to a certain sampling frequency is dynamic. Based on these two meanings, reasonable communication middleware can not be ignored in the distributed scene. On the one hand, middleware can weaken the problem of heterogeneous sensor nodes caused by physical distribution. On the other hand, middleware can reduce the difficulty of using dynamic data. Common methods include preprocessing raw data and integrating historical data.

Contemporary logistics sensor platforms are highly heterogeneous. Platforms offer a wide range of sensing, computing and networking facilities at the hardware level. At the software level, sensor nodes run various operating systems and middleware [29]. Current logistics sensor platforms are also highly distributed with third-party sensing infrastructure deployed in geographically distributed trucks, trailers and warehouses that should come together as required to enact a sensing application [57][20]. Finally, sensing application requirements are also dynamic. A single WSN can support multiple sensing applications, such as location tracking, cold-chain monitoring and/or ensuring the safe storage of hazardous chemicals.

Development of middleware techniques for WSN is motivated by two factors: on the hardware aspect, data processing ability on sensors is dramatically improved; on the software aspect, increasing scale of applications are developed which require to be supported by heterogeneous platforms. Therefore, as a software layer between operating systems and distributed applications, WSN middleware that provides the mechanism of interoperating among distributed software modules should be able to shield differences on physical layer architecture and communication architecture.

By learning classic cases [14][15][55] and reviewing relevant literature, WSN middleware classification based on underlying programming approaches could be grouped as a virtual machine [6], modular programming [9], database [37], application-driven , and message-oriented approaches. Also, middleware could be grouped based on their design approaches as agent-based [22], cluster-based, component-based [58], and QoS-based. However, according to a survey from Mohammad [55], there is no single existing middleware that can consider all WSN system requirements as well as the solution to all limitations of WSNs.

One of the most crucial challenges for middleware design is how to deal with trade-offs such as the trade-off between energy consumption on data processing upon local sensor nodes and data transmitting among sensor nodes. Besides resource discovery for dynamic applications and code management, it cannot be solved well in existing middleware approaches. When dealing with a trade-off mentioned above or the conflict occurred during resource management, a reasonable approach of resource allocation by the middleware is essential.

2.3 Edge Computing

From the perspective of intelligent logistics, the warehouse is one of the data centres that generate massive interlaced and correlative data. For general warehouse management, there are two application uses of sensed data: one is for cargo management, including goods identification (using RFID) and goods tracking (location and movement); the other is for safety management, which refers to environmental monitoring and data security. It is practical to filter and process data rapidly for applications requiring a fast response and real-time tracking [52], such as those involving safety hazards [31]. Some existing warehouse applications enable safety hazards to be detected only by visualising raw data and human response at the proximity of the users and rely upon the higher-level computational ability and strategy-making ability of the cloud for prediction. The resource limitations of WSN infrastructures restrict the state-of-the-art paradigms such as deep learning, which requires large capacities in terms of computing and storage, to be implemented on terminal nodes [47]. Such a WSN-plus-cloud mode causes problems that lead to either high bandwidth usage or high latency in undertaking emergency interventions. A well-designed WSN-Edge-Cloud system architecture [48] would be a suitable solution when facing the requirements of a “rapid response” system on both WSN localisation and cloud globalisation which cannot be satisfied by a simple WSN-Cloud architecture. A WSN-Edge-Cloud system integrates edge computing features with the WSN-Cloud architecture to solve the problems mentioned above to improve the efficiency of business logistics.

Known as a new paradigm for the IoT domain, edge computing [21] moves complex computation from the cloud to the edge of the networks and devices. Edge computing does not aim to replace cloud computing but instead focuses on balancing latency of the communications and accurate computation for the wider IoT hierarchy [10][12][32]. In a WSN-Edge-Cloud system architecture, edge computing can be regarded as a relay between the cloud and sensor nodes in the WSN [54]. The edge computing nodes extend the cloud computing paradigm to the edge of the network in a bidirectional way [27]. Edge nodes focus on local functionality to support geographically closer sensing on the node-to-cloud direction with the additional feature of data pre-processing and rapid reaction time. According to application requirements, this aggregated, filtered, and pre-processed data is sent to the cloud selectively. In the cloud-to-node direction, edge nodes achieve distributed deployment of the broad class of applications under the macro-control of the cloud and perform the tasks allocated by the cloud. Since edge computing reduces both the response time in IoT communications and the upload bandwidth to the cloud, the resulting applications will have the features of real-time interaction at the edge, and prediction analysis in the cloud [53].

2.4 Mobile Agent

As presented in several research works [19] [41] [24] [59], with the development of WSN techniques and distributed network algorithms, a network's primary requirement is no longer the query of resources and services from local hosts or traditional computers; instead, the optimisation of resource consumption and services is the focus.

With performance improvements in system efficiency, scalability, as well as mobility, the concept of Mobile Agent (MA) [14][68] becomes one of the crucial techniques that aim to support the easy deployment of applications on sensor nodes in a WSN system[49][26]. Mobile Agent is a kind of computer software, but mobile agent's networking character of mobile agents lets it be separate from the computer system [60]. Adhering to individual specific migration strategy among the heterogeneous networks, mobile agents aim to search three core resources required to solve network problems: computing resources, information resources, and software resources [7]. The obtained resources are used to process data and achieve specific work in a particular approach. Its data processing ability decides that WSN system with MA can significantly extend the lifespan of terminal resources sensor nodes by reducing the energy consumption of data processing on local sensor nodes. An

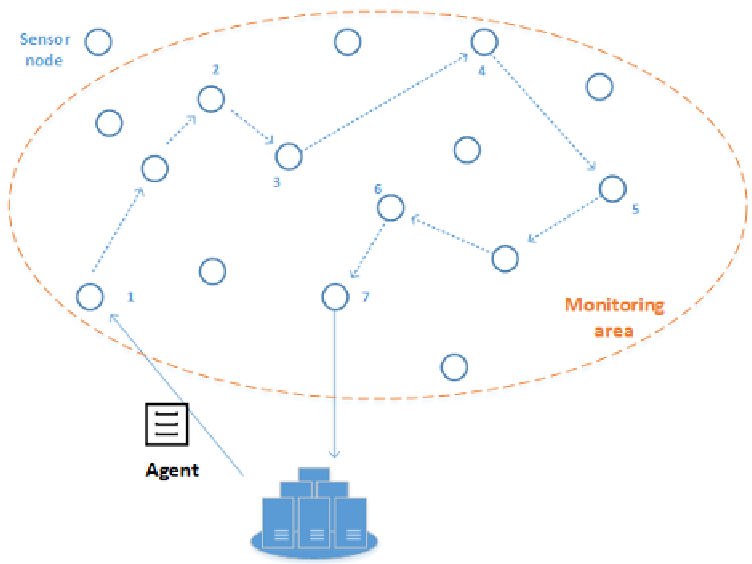


Figure 2.1: An example of WSN with agent system applied in. The little circles denote sensor nodes while the large ellipse in dash line denotes agent migration order. Arrows denote one hope of agents among sensor nodes.

example of a sensor network that involved agents is shown in Figure 2.1. The little circles in Figure 2.1 denote sensor nodes while the large ellipse in dash line denotes monitoring area. Numbered sensor nodes denote agent migration order and arrows denote one hop of agents among sensor nodes respectively.

The mobile agent technique proposed in this thesis is a combination of traditional agents and distributed computing techniques. Without the requirement of centralised dispatch, mobile agents have high autonomy and intelligence [34]. They are able to operate on heterogeneous sensor nodes and give feedback after data processing. Besides, more than one agent can cooperate to achieve a specific task in multiple agent systems by operating on several sensor nodes synchronously.

Although various MA algorithms, mechanisms and optimisations described in the literature mentioned above, there is a lack of systematic explanation of the cooperation mechanism between multiple MAs considering the character of highly autonomous WSN. Moreover, these MA systems are typically based on a relatively independent system or middleware platform that does not maximise the application of existing node infrastructure.

2.5 Chapter Summary

The research background is summarised into four aspects in this chapter, which are (1) the worth of Wireless Sensor Network (WSN) in the domain of industry and logistics; (2) the existing middleware platforms especially those fit for WSN application scenarios and the emergent demand on the middleware techniques that support for dynamic applications; (3) the definition and features of edge computing and the value of adopting edge computing to the WSN application scenarios; and (4) an introduction of mobile agent technique and its value on the dynamical applications.

Chapter 3

Architecture

As described in Chapter 1, this dissertation focuses on the “Edge Computing (EC)” and “Mobile Agent (MA)” techniques in order to overcome the challenges of low latency and fast response requirements of the dynamic WSN applications in the sensor cluster environment with limited computing resources. EC and MA, as the cutting-edge technologies in WSN communication middleware, need to rely on a reasonably designed top-level architecture. Architecture design includes methodology design, composition design, algorithm design, etc. This chapter focuses on the architecture throughout the research process of the dissertation. The specific EC and MA research outcomes to be introduced in Chapter 4 and Chapter 5 are based on the architecture illustrated in this chapter.

3.1 Hierarchical Edge Computing Structure

A rapid response system architecture for the distributed management of warehouses in logistics by applying the concept of tiered edge computing is illustrated. A tiered edge node architecture is proposed for the system to process computing tasks of different complexity.

Compared with classical WSN system architectures, the proposed edge computing-based hierarchical system architecture consists of three core layers, as illustrated in Figure 3.1. “EN” indicates edge node. A smaller EN-Grade number indicates the physically closer location from the edge node to the sensor nodes and sensing devices. As a widely accepted environmental sensing infrastructure, sensor nodes in the WSN collect sensing data and track changes to the environment continuously. For a better identification and management, sensor nodes in the WSN are logically separated into different areas. By

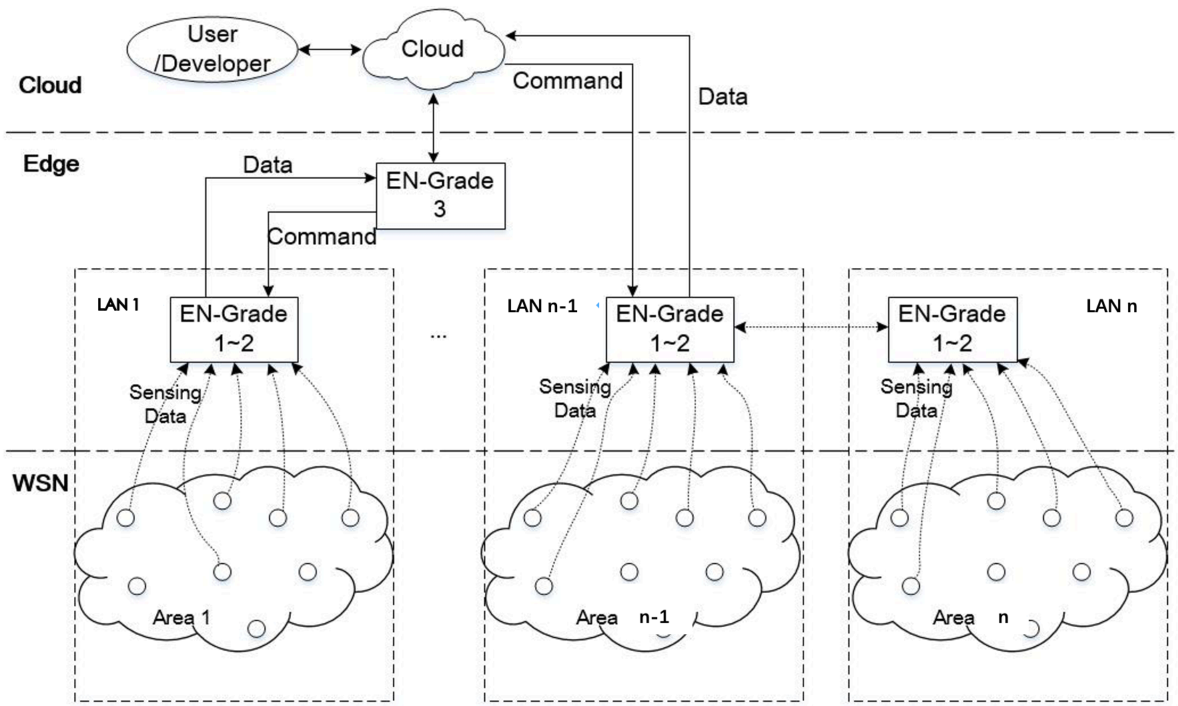


Figure 3.1: Edge-computing-based tiered system architecture. "EN" indicates edge node. A smaller EN-Grade number indicates the physically closer location from the edge node to the sensor nodes and sensing devices.

moving the computation to the edge of the networks, the edge computing layer reduces the response time during communications as well as the required upload bandwidth between the sensor networks and the cloud.

Compared to just using the cloud, the edge layer is physically close to the environment. The functionality of the edge layer in the proposed structure is refined into three grades of edge nodes. Grade one and two edge nodes are focused on general functions including data formatting, preliminary data processing for WSN data collection, and the execution of tasks and control commands allocated by the upper layer (higher grade edge nodes or the cloud). Grade three edge nodes contribute to more complex data analysis, which involves data that is potentially useful for prediction and control, as well as generating or relaying control commands from the upper layer down to the lower layer. Cloud computing is docked to the cloud layer, contributing to the centralised analysis of global data and the management of the entire network. In addition, the connection between users and the system via the cloud realises the remote operation and control of all areas covered by the terminal devices. For application developers, the system can be accessed via the cloud or edge node for application deployment depending on the deployment requirements and the network condition.

3.2 Middleware platform support for MA

3.2.1 Requirement analysis

The requirements for the middleware framework is identified as follows in order to realise the objective (a) as introduced in Chapter 1, which is a middleware technology that supports dynamic real-time WSN tasks by achieving the cooperation between sensor nodes.

Energy consumption: Considering that the battery of a general low-cost sensor node has a limited life and is non-replaceable, the energy consumption of the nodes applying the target system is crucial. According to the purpose of nodes in a WSN system, energy consumption should be distinguished. For example, the gateway or the sink-nodes should be more potent than the standard sensor or the actuator nodes.

Resource limitation: Being similar to the battery life of a general sensor node, the memory resource and computing ability are generally weak as well. The system should take these resource limitations into account.

Scalability: Limited battery life and unforeseeable damage of the sensor nodes indicate the high scalability requirement for a WSN. It is necessary to add new nodes into a WSN quickly and consistently.

Reliability: The system and network should have the ability to tolerate perturbations that might affect the system's functional body.

3.2.2 MA-based middleware framework

From a software-hardware perspective, a WSN system consists of a large number of physically distinct nodes and clients with functionalities of sensing environment, processing data, and communication. The functions collaborate to complete the WSN applications required by users.

The logical layers of one sensor node consist of three basics: application layer, cross-layered middleware modules, and an underlying operating system layer. Different types of multiple MAs belong to the application layer. Their role is to independently or collaboratively complete the application tasks used to be pre-written into the node. Compared to traditional integrated applications, application tasks implemented by MA have significant mobility characteristics. Different applications can be deployed in real-time according to user requirements, which has good benefits in eliminating the process of pre-set applications and enhancing the general characteristics of the network to some extent.

According to the functionalities that can support MA working on nodes, the core modules in the cross-layered middleware include a MA manager, a network manager, and a local parameters manager. The MA manager processes transactions related to the MAs.

By parsing the specific segments in an MA message, the manager splits the MAs in different types and with different functions. Based on the parsed agent type, the manager determines the migration strategy for an MA while processes the task code carried by it. For MAs with the unique identifier that indicates a requirement of cooperation, the manager will authorise through the MA collaboration mechanism. The network management module maintains a local neighbour list on a node. The maintenance covers both initialisations of the list and updating of new neighbour nodes or expired nodes. From the local parameter manager, a MA can acquire the local parameters such as node ID, the battery life of the node, sensing data, etc. To illustrate the logical layers described above clearly, the left-hand side in Figure 3.2 shows the overview architecture of the node with the MA-based system. Corresponding to the logical layers, the right-hand side part in Figure 3.2 shows the physical structures which are the user, MA middleware platform, and node infrastructure.

WSN middleware, which is often developed in the form of a cross-layer architecture [55], is designed to hide complex underlying hardware environments and cumbersome application deployment processes from distributed application developers. The state-of-the-art MA-based WSN middleware [18, 8]usually provides its unique instruction set and MA interface to the developers by integrating the complex underlying operations and adding the directional service contents.

As shown in Figure 3.2, in addition to the primary network management module and topology management module, the MA management module is also included. These manager modules provide a convenient infrastructure for MA-based applications through cross-layer collaboration. In order to reduce development costs, the middleware platform referred to our work presented in [63] focuses on the CoAP protocol, based on which we can support MA applications that conform to the CoAP message format. At this point, MA-based middleware can provide complete platform support for application development

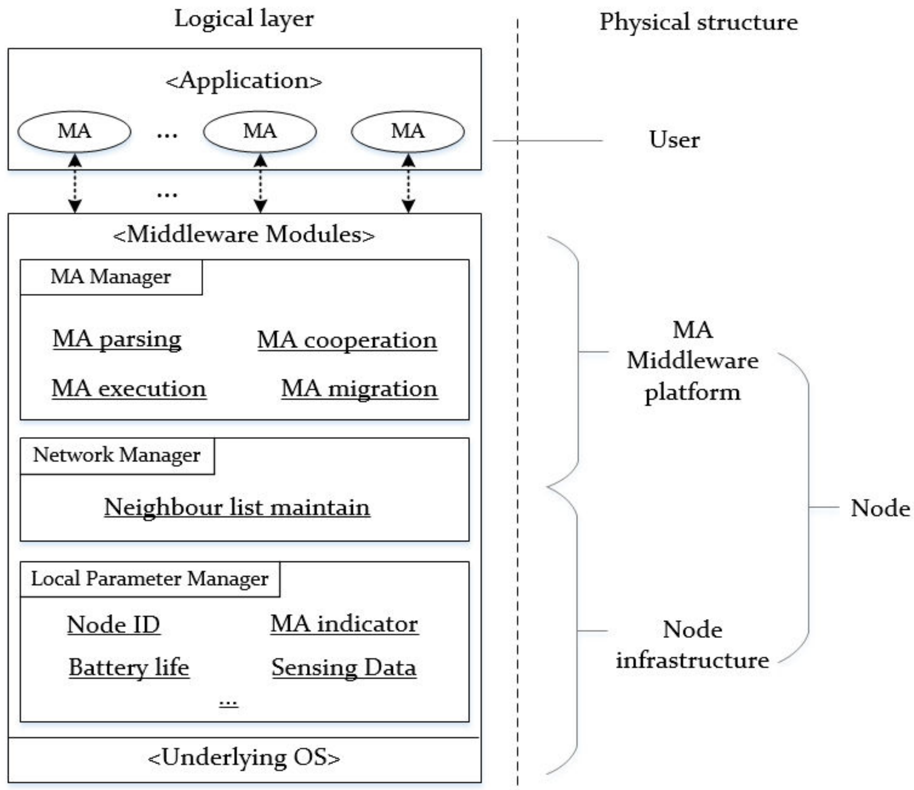


Figure 3.2: Overview framework of the MA-based middleware. The left-hand side shows logical layers, which includes application layer, cross-layered middleware modules and underlying OS. The corresponded physical structures are shown on the right-hand side.

and deployment for real-time needs. In the future development process, compatibility with heterogeneous nodes can also be extended.

3.3 Chapter Summary

In this chapter, we have proposed a hierarchical structure for the edge layer in the WSN-Edge-Cloud architecture to meet rapid response requirements in the intelligent warehouse scenario. The rapid response indicates that by taking advantage of the edge computing, the system gives the first-time response to exceptions near the data-generating location rather than waiting for the response strategies from cloud computing, to improve the response speed of the system. Instead of the standard approach of an integrated single edge layer in the WSN-Edge-Cloud architecture, this chapter proposes a novel approach of a hierarchical edge layer with cooperated edge nodes in each of the different tiers. In this hierarchical structure, the edge functions are modularised and abstracted according to the coupling degree of functions and applications. General functions loosely coupled with concrete applications such as data formatting are achieved on low-level edge nodes while the application-specific functions are allocated to high-level edge nodes. Such grading strategy keeps the advantages of edge computing with low latency and enhances the reconfiguration of hierarchical modules at the edge of the network.

A mobile agent-based middleware framework designed for WSN is proposed in this chapter. The core idea of the middleware is to optimise the allocation of resources while taking the compatibility of the heterogeneous platform into account. Combined with the mobile agent cooperation strategy we have proposed, the inventive mobile agent-based middleware framework could optimally achieve the intensive requirements of a WSN application. The middleware has considered resources limitation issues commonly addressed on normal sensor nodes. Meanwhile, it provides a possibility of platform independence as well as programming language independence. It can also reduce network payload and dynamically adapt to environmental changes.

Chapter 4

Hierarchical Edge Computing

This chapter presents a rapid response system architecture for the distributed management of warehouses in logistics by applying the concept of tiered edge computing. A tiered edge node architecture is proposed for the system to process computing tasks of different complexity, and a corresponding rapid response algorithm is introduced. The classification of abstracted outlier sensing data is emphasised, which could better match different sensing types and transplant to various application fields. A software-defined simulation evaluates the system performance on response time and response accuracy.

4.1 Methodology

4.1.1 Distributed Micro-Database

The architecture of the rapid response system is presented in the Section 3.1 of Chapter 3. As stated, grade one and two edge nodes in the edge layer focus on general functions such as data formatting and preliminary data processing for WSN data collection. A lightweight database (SQLite [4]) component is inserted between the grad-1 and grad-2 edge nodes in the system. The purposes of including a light-weight database are (a) capturing the data effectively, (b) supporting selective data retrieval, (c) filtering data without additional programming, and (d) enhancing data readability and translatability. The interaction between grade one and two edge nodes is illustrated in Figure 4.1.

Compared to some popular databases such as MySQL [1] and PostgreSQL [3], SQLite is a good choice as it has low resource usage suitable for use in embedded products for IoT ap-

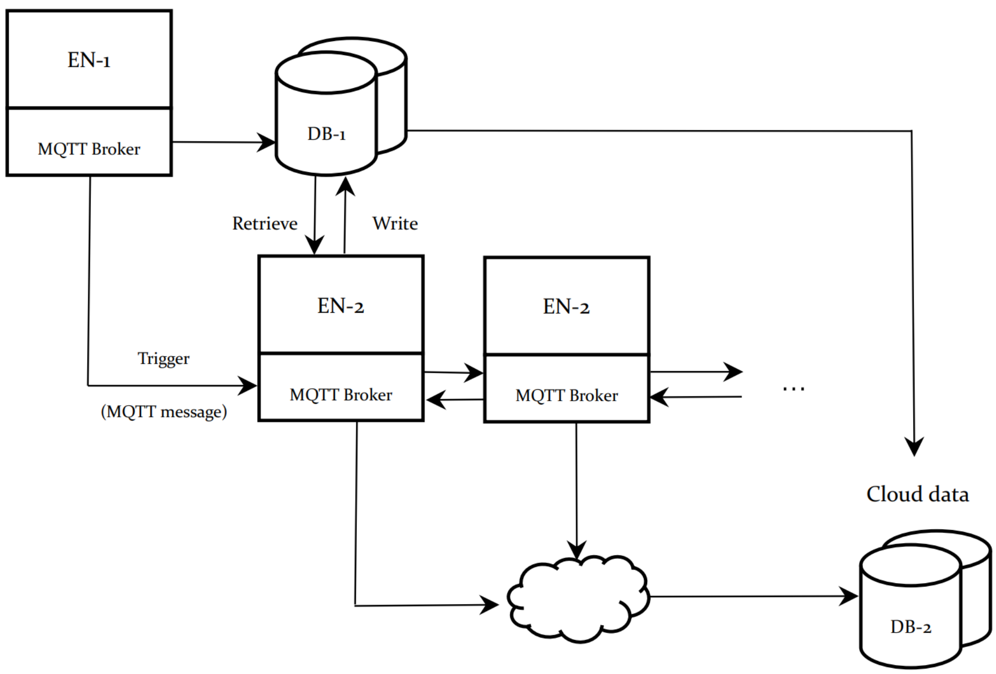


Figure 4.1: The interaction between grade one and two edge computing nodes in the architecture for WSN.

plications. Due to communication dependencies on the database, the main communication protocols are called via APIs within the program directly. SQLite has outstanding performance, low-resource consumption, low latency, and overall configuration and management simplicity.

As a common agent component in the WSN communication environment, message brokers that implement communication protocol would deal with messages from endpoints that generate massive data. The communication between EN-1 and EN-2 shown in Figure 4.1 is achieved by implementing message brokers on all the edge nodes. Since most IoT communication brokers such as Message Queuing Telemetry Transport (MQTT) [50] brokers do not provide any mechanism to log historical data, a script to log sensed data to SQLite is written based on python benefitting from the multiple programming language adaptation feature of SQLite. The script will log data on a collection of topics, includeing message time, message topic, and message payload. Considering the sensed data has high repeatability, the script only logs changed data from the status sensor, indicating that if a status sensor sends its status as "ON" once per second, it could result in 3600 times of "ON" messages logged every hour. However, the script only logs one message. The script uses the main thread to get the data (on message callback) and a worker thread to log the data. A queue is used to move the messages between threads. Once the data is placed in the queue, the worker will take it from the queue and log it into a disk. The worker is started at the beginning of the script.

4.1.2 Abnormal Data Type Abstract

At the initial stage of the experiments on the research related to the hierarchical edge computing as presented in [66], four abnormal sensor data types were defined for the rapid response system testing in the warehouse scenario, which included (a) rapid-growth, (b) slow-growth-diffusion, (c) slow-growth-nondiffusion, and (d) error data on a single node. The classification was based on the data change rate and considered the diffusivity of the sensed object.

However, to loosely couple the outlier detection approach with the specific sensing data type, the abnormal data types are summarised into three abstracts that aim to improve the portability of the system for adapting to various application scenarios. An enhanced outlier detection mechanism is proposed to support such abstracts. The following section presents the outlier type abstracts and related detection mechanisms.

Sensing information collected by sensors is classified into two forms, (a) numerical data and (b) status data. In the detection and rapid response system, our monitoring method for numerical data anomalies is mainly to record comparison results, while the detection method for state data anomalies is to record state changes. The detection of numerical data is commonly based on a certain preset threshold, specifying whether it is an outlier. Two methods are considered for setting a threshold: constant value and prediction formula. The constant value is applied for data with high stability, such as temperature data from an incubator. Prediction formula, which requires preset formula or complex prediction calculations, is often used for sensor data with uncertain change trends and large overall fluctuations. The monitoring data sensed based on the road vehicle flow is a typical example. Besides exceeding the threshold, data jitter within the boundary threshold is considered as a type of outlier as well. The core performance of data jitter is the sharp increase/decrease in the dispersion of values, i.e., the degree of deviation of the sensing data from the average value increases under the premise of the constant sampling interval. In the domain of WSN, sensor data has a strong relation with time series. Hence, the processing of sensor data needs to consider both the timing factor and the data volume. Considering the premise that the sensor nodes have limited computing capacity and limited storage space, we simplify the calculation of the real-time standard deviation of data jitter into the calculation of data increments. We set a threshold for the increments, thereby transforming data jitter anomalies into threshold anomalies.

As for the status data, an expect-states pattern, which holds all effective statuses of the sensor data, is used for the outlier detection. One of the typical status sensors is the bi-state switch. In an application scenario where the bi-state switch is required to be naturally 'ON', the expect-states pattern will hold the 'ON' state only, which indicates the 'OFF' state is an outlier.

Based on the three outlier data abstracts presented above, a rapid response strategy for the edge nodes in the system to justify the abnormal types is proposed in the following section.

4.1.3 Rapid Response Strategy

Within a target monitoring area, there are two primary cases in which sensor nodes are supposed to generate abnormal sensing data: sudden environmental change and error data caused by a broken sensor or irruption. A rapid response is only expected to be triggered by

Table 4.1: Outlier type abstracts and specifications.

Outlier types	**Quantitative Reference**	**Allocated EN**
Exceeding threshold	Constant value	EN-1
	Prediction formula	EN-2 and u-Database
Jitter	Real-time standard deviation	EN-3 and u-Database
	Increment	EN-2 and u-Database
Status change	Status pattern	EN-1

the first case, which could save time for emergency interventions and reduce the potential for business losses. In contrast, a rapid response caused by the second case will lead to a waste of resources. The strategy aims to initially process the data near the edge, at the data generation end, quickly identify the types of abnormal data, and make response strategies. It avoids the system being overly sensitive caused by responses to the raw data. Meanwhile, it could reduce the waiting time on uploading redundant data to the cloud as well as the waiting time on the instructions from the centre server.

Table 4.1 lists the proposed outlier type abstracts and specifications. With the hierarchical edge computing architecture illustrated in this chapter, we distribute three abstracts of outlier type detection to different tiers of EN. Based on the principle that the lower the EN's grade, the lower the computing complexity, EN-1 detects the status changes and numerical data that exceed the threshold with a constant value. EN-2 processes the data exceeding the threshold with a prediction formula in coherence with historical data retrieved from the local micro database. The computing that involves real-time standard deviation based on massive data and related prediction will be allocated to the higher tiers of EN even to the Cloud. The detailed process is proposed as pseudo-codes in Algorithm 1, where r indicates compare result in both cases. Since thresholds may either be upper boundary or lower boundary, we use Boolean variable r to denote whether data value exceeds the boundary. Data gradient set is defined as:

$$Grad(n) = grad(n, i), n \subseteq N, i : timestamp \tag{4.1}$$

Data gradient is defined as:

$$grad(n, i) = \frac{d(n, i + T_c) - d(n, i)}{T_c}, n \subseteq N, i : timestamp. \tag{4.2}$$

Algorithm 1 Rapid response strategy

Input: Status pattern set: SP,
  Threshold: TH,
  Prediction formula: f,
  Data set: D(n)=d(n,i,tp,payload),n$\subseteq$ N,i:timestamp,tp:datatype,
  Node set: N=n_1,n_2,...,n_k,k:number of nodes in the area

Output: Process on EN-1:

detect d(n,i,tp,payload)
write(d,Database)
if tp == "status" **then**
  $rs \leftarrow compare(payload, SP)$
else
  $rt \leftarrow compare(payload, TH)$
end if
if rs == **true** **then**
  print status changed
end if
if rt == **true** **then**
  print exceeding threshold
  generate M=(n,i)
  activate EN-2
end if
Process on EN-2:
observe M
retrieve (D_{nb}, Dataset)
$r \leftarrow compare(f(m), TH)$
if r == **true** **then**
  print exceeding threshold
end if
$r \leftarrow \frac{\#\{grad(m,i)>0\}}{\#\{N_{nb}\}}$
if r == **true** **then**
  print jitter
else
  activate EN-3
end if

4.2 Implementation

In this section, the proposed method is evaluated using software-defined networks. The platform, hardware settings, and response accuracy are reported.

4.2.1 Implementation Platform and Hardware

Considering that massive sensor nodes in IoT applications cannot be completely simulated in a laboratory scenario for research experiments, we selecte an open-source flow-based tool and platform called Node-RED [2] for implementing the architecture proposed in this chapter. Node-RED is used as a platform to integrate components at multiple layers in the network. It provides a convenient connection with the web interface to visualize the data flow and configure the network. In our simulation, ENs communicate by subscribing to a selected topic on a broker, and the messages/data coming into the topic could be observed on the web interface. The Node-RED is run on both the Raspberry Pi (Rpi) machines and laptops as the initial setup of the network. During the experiment, there are three PCs including two laptops and one workstation, two Raspberry Pis and a dozen of Arduino boards are used to setup the hardware environment. The number of the equipment is decided by the proposed system architecture as shown in Figure 3.1. Both laptops and Raspberry Pis are installed Node-RED to achieve the higher graded edge nodes' functionalities. With Node-Red running on the equipment, we could manage and configure the network and component connection using a browser. Thus, all the platform and hardware are set up as listed in Table 4.2. Besides the physical sensors, python scripts are used to simulate sensing data for various scenarios that currently cannot be implemented in the laboratory.

As shown in Table 4.2, Raspberry Pi and PC/laptops are used as edge nodes and the MQTT broker, as well as CoAP MA, are deployed on edge. Users could access the web-based client to retrieve raw data and preprocessed data. The network for the whole system is simulated based on Node-RED simulator. The cloud is planned to apply IBM open-source cloud as some of the IoT application components are integrated and free to use. The block diagram in Figure 4.2 shows the architecture of the simulation environment, which contained the components listed in Table 4.2. Figure 4.3 illustrates an example Node-red flow that implements the functions of EN-1.

Table 4.2: Simulation platform and hardware.

Implementation	Networking (Functionalities)
SN, AN* : (a) Arduino board (b) Python code (simulation)	Data collection Data formatting (JSON)
EN* : (a) MQTT broker (on Rpi, PC) (b) CoAP MA (on PC)	Response tasks: Modularised Reconfigurable Cooperative
Web-based Client: (a) MQTT Lens (b) CoAP client	Data visulisation
Network: (a) Node-RED simulator	-
Cloud: (a) IBM cloud (to do)	-

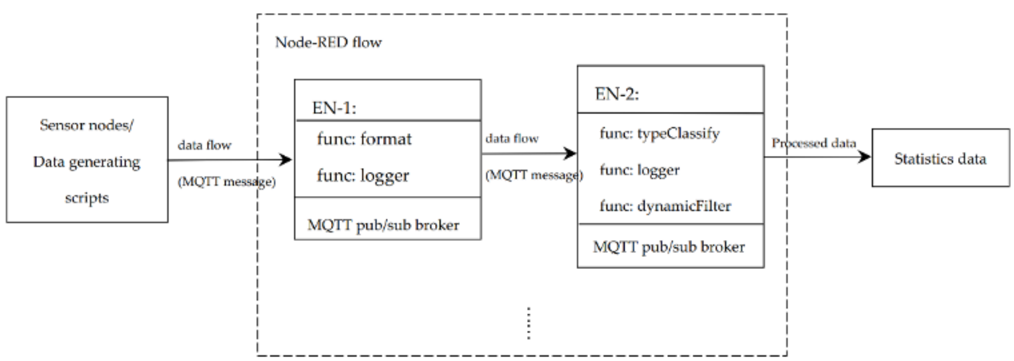


Figure 4.2: Architecture of the simulation environment.

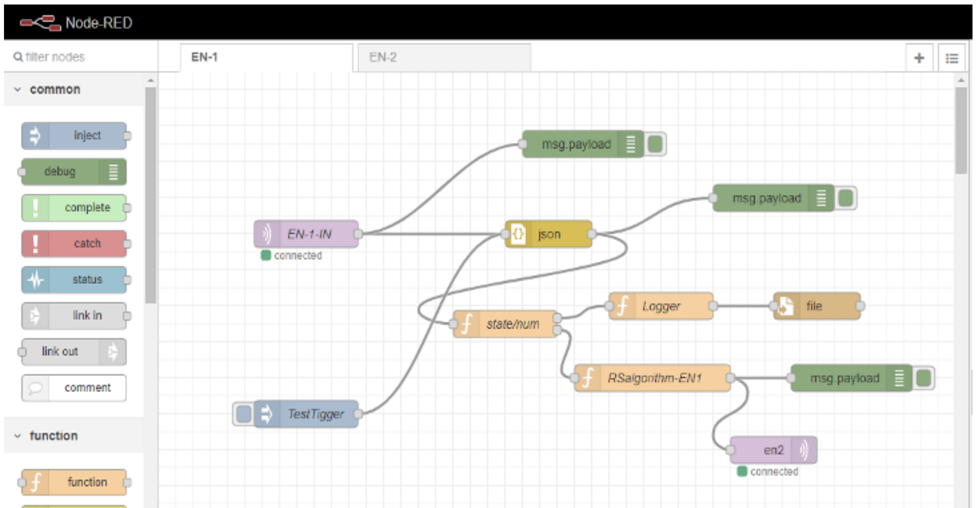


Figure 4.3: Example Node-RED flow that implements the functions of EN-1.

4.2.2 Case Study Model

The experiment scenario is a cognitive warehouse which could recognise and respond to unexpected cases as illustrated in Table 4.1 rapidly. The temperature is selected as the sensing data type during the case study. The sampling rate of sensor data is configured as once per ten seconds, once per thirty seconds, and once per minute for three groups of repeating tests. The network scale is configured as 1000 nodes, 5000 nodes and 10,000 nodes, respectively.

When setting the network scale as 1000 nodes, the nodes are grouped into 50 LANs which keep generating sensed data. Every 20 sensor nodes in the same group are considered as neighbours and connected with EN in a Star topology. Besides the normal data (set as 26 $\pm$ 1 centigrade for initialization), three types of outlier data are inserted into the data set randomly, which include over the boundary (constant and predicted), jitter, and error. Figure 4.4- 4.7 illustrate four example cases during our experiments with a fixed sampling rate as 10 per second and a fixed network scale as 1000 nodes. Figure 4.4- 4.6 judge outliers based on the constant threshold. The difference between Figure 4.4 and Figure 4.5 is the duration that outliers been detected. These cases would be reported as "exceeding threshold" while the case in Figure 4.5 needs to be further processed by EN-2. The case shown in Figure 4.6 indicates that jitter happened to the sensor node. The single node's value does not exceed preset boundaries, but it brokes the stability of the sensed

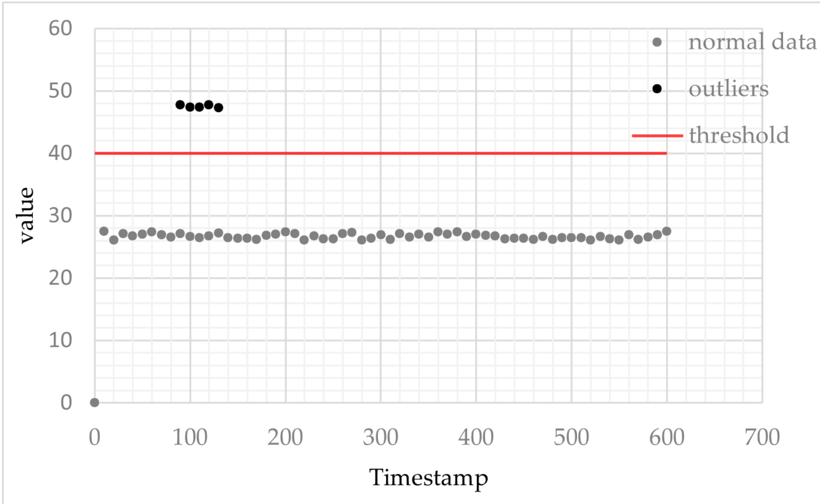


Figure 4.4: Outliers exceed a constant threshold in a short period

data, which could be considered as one of the general outlier types. The case shown in Figure 4.7 denotes the threshold based on the prediction formula. Data which deviated from the expected scope would be considered as outliers.

Such data flows are tested under three different network scales with two different sampling rates by the system. The outcomes and related analysis are presented in the next section.

4.2.3 Outcomes and Analysis

We only test and compare the numerical sensor data during the experiment to normalise the analysis. Instead of practical environmental data, our dataset is designed for testing the proposed architecture and algorithm performances in the simulation. During the test under the setting of a 1000-node network scale and 0.1sec sampling rate, 50 groups of data are generated and tested corresponding to the WSN in 50 LANs. Packs of sensor nodes for each corresponding edge node are configured as 20, which is a fixed parameter during the whole experiment. In other words, every 20 sensor nodes in the same group are considered as neighbours that shared data at their corresponding edge node. Four types of numerical outliers are inserted into a different dataset for each group, which are "error", "exceed threshold", "jitter", as well as "dynamic threshold". The system average response time and accuracy are used to reflect the system performance. The system response time

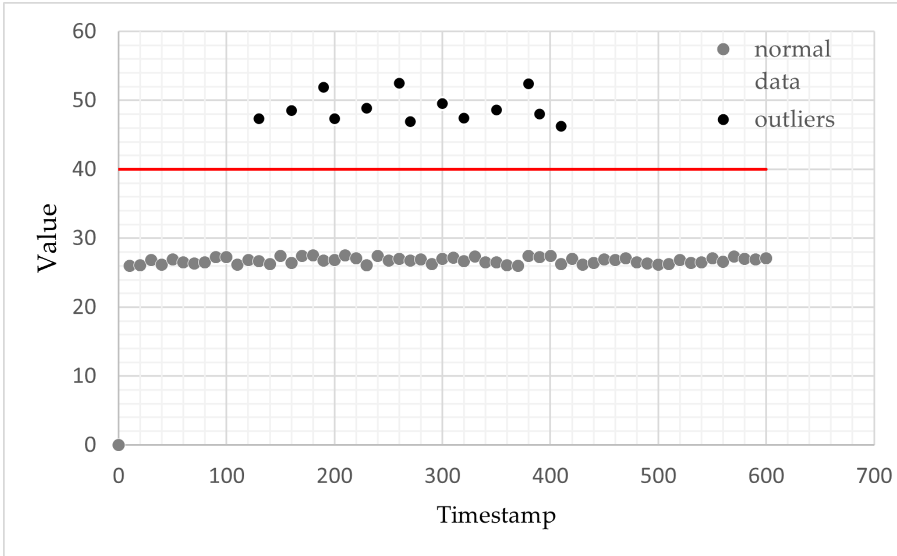


Figure 4.5: Outliers exceed a constant threshold in a long period

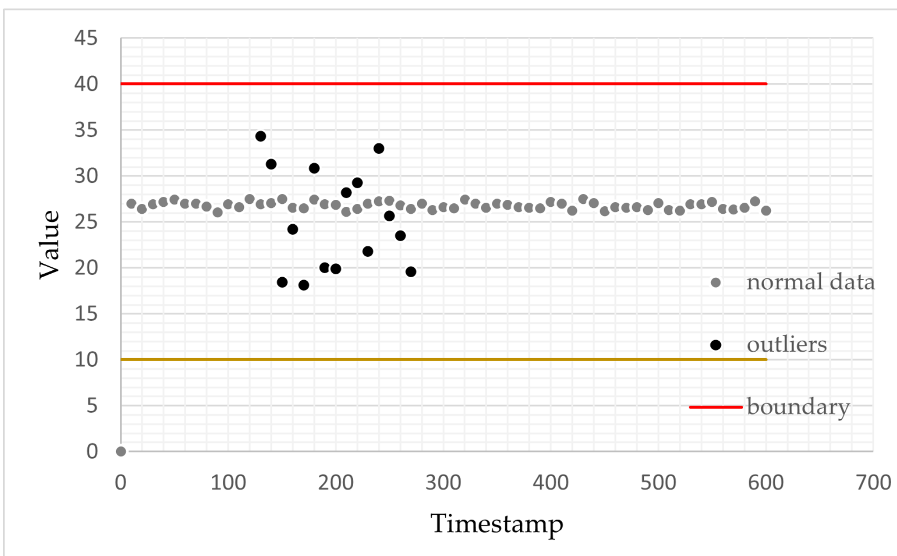


Figure 4.6: Outliers act as jitter within boundary threshold

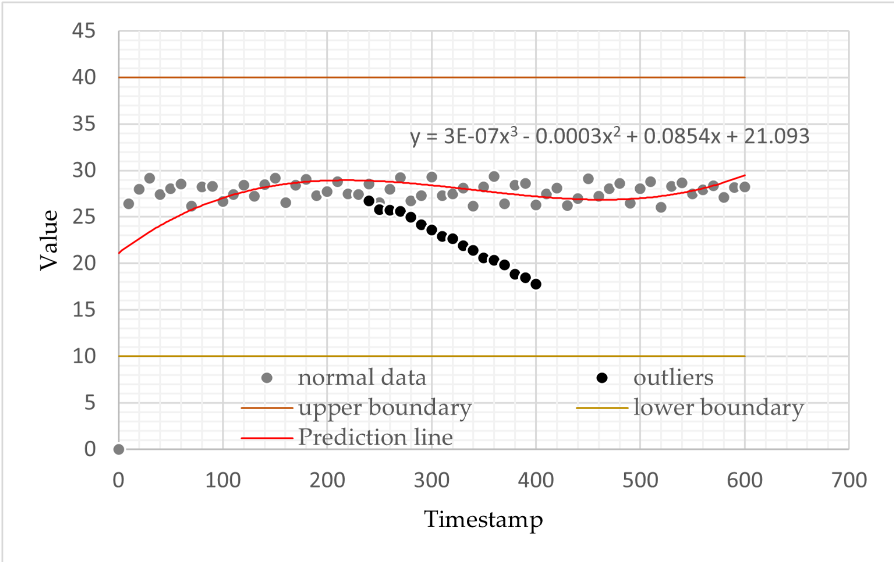


Figure 4.7: Outliers deviate prediction line

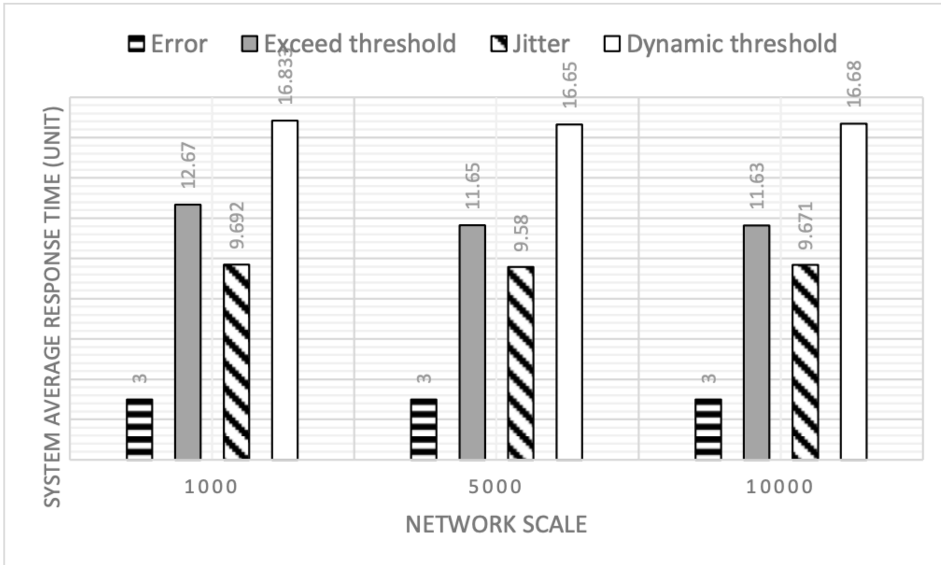
is calculated by:

$$\frac{T_{response} - T_{detect}}{f_{sampling}} \tag{4.3}$$

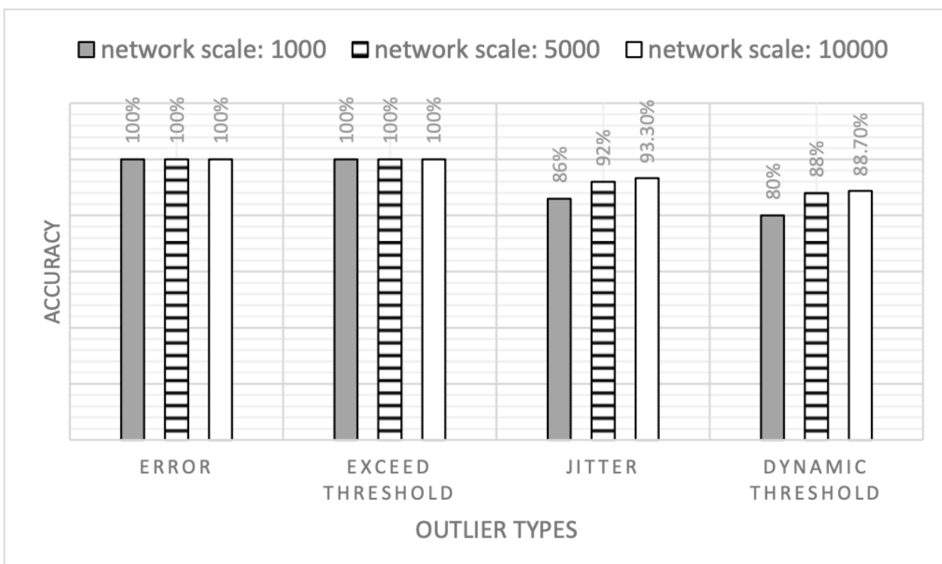
where $T_{response}$ denotes the timestamp that the system respond to the outlier, T_{detect} denotes the timestamp that the outlier is detected, and $f_{sampling}$ denotes the sampling rate of the sensed data. The accuracy is calculated by dividing the number of outliers that are detected correctly by the number of original datasets on each end node.

The comparison of the statistical testing results with different parameters are shown in Figure 4.8 and Figure 4.9. The original data are shown in the tables in Table 4.3-4.6.

By analysing the statistical outcomes shown in Figure 4.8 and 4.9, it can be observed that cases related to the errors and exceeding the threshold could be 100% detected by the system. However, the system average response time has a noticeable difference. The results would not be affected by the network scale and the sampling rate. As introduced in the methodology section, any data process that require retrieving historical data from the local database need to be handled on EN-2. Therefore, extra data retrieval time and increment calculation lead to a relatively long system average response time for the exceed threshold cases. The cases relating to the jitter have above eighty per cent accuracy with a 9.69 sampling unit average response time which indicates that the edge system could

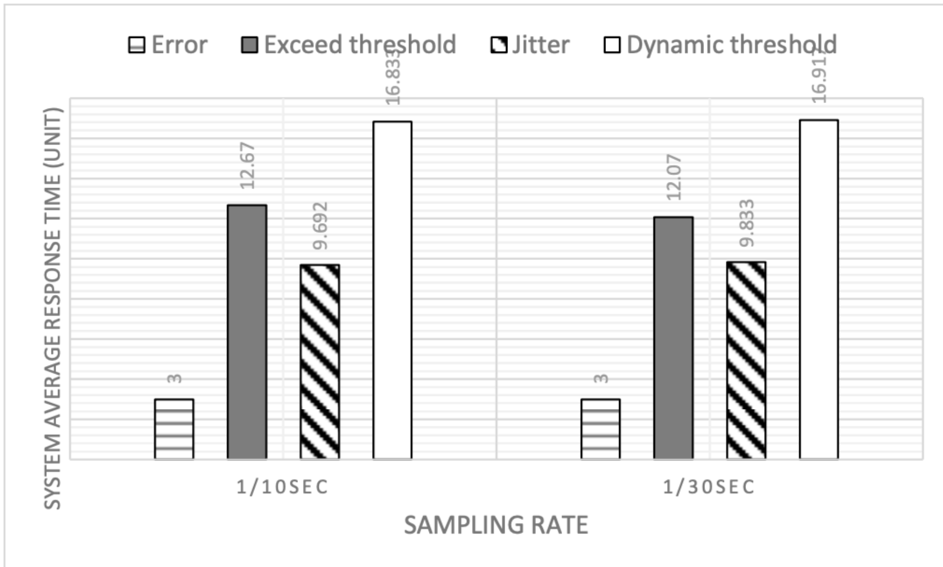


(a)

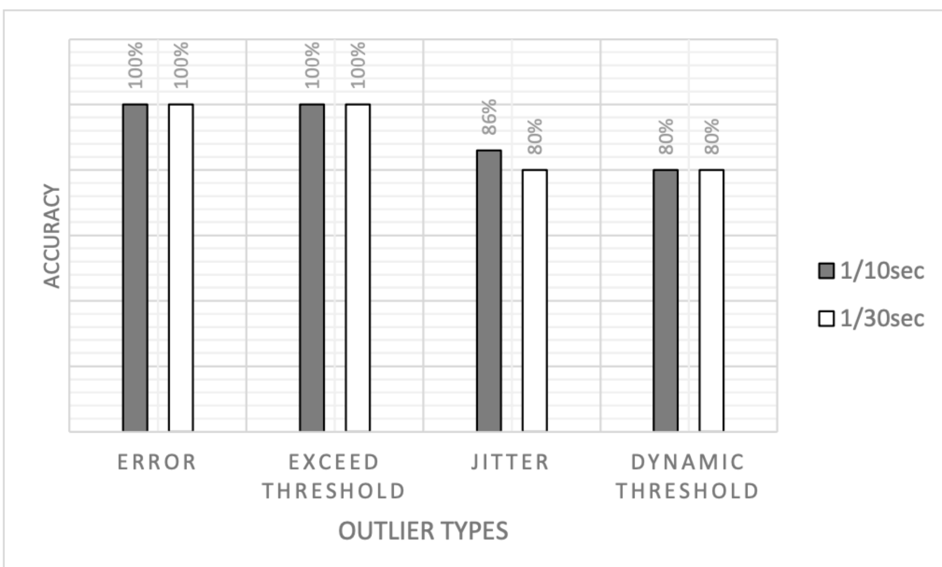


(b)

Figure 4.8: The comparison of the system average response time and accuracy under different network scales and fixed sampling rate (sampling rate = 0.1sec)



(a)



(b)

Figure 4.9: The comparison of the system average response time and accuracy under different sampling rates and fixed network scale (network scale = 1000)

Table 4.3: Statistical outcomes of the experiment (sampling rate = 1/10sec, network scale = 1000 nodes)

Outlier Types	**Number of Original Data Sets**	**Number of Outliers Being Detected Correctly**	**System Average Response Time (Unit)**	**Accuracy**
Error	5	5	3	100%
Exceed threshold	15	15	12.67	100%
Jitter	15	13	9.692	86%
Dynamic threshold	15	12	16.833	80%

Table 4.4: Statistical outcomes of the experiment (sampling rate = 1/10sec, network scale = 5000 nodes)

Outlier Types	**Number of Original Data Sets**	**Number of Outliers Being Detected Correctly**	**System Average Response Time (Unit)**	**Accuracy**
Error	25	25	3	100%
Exceed threshold	75	75	11.65	100%
Jitter	75	69	9.58	92%
Dynamic threshold	75	66	16.65	88%

detect the jitter-type data exception successfully within ten sampling units. Eighty per cent of the tested outliers defined by the dynamic threshold could be detected by the edge system within 17 sampling units. The sampling interval does not impact the system average response time and the response accuracy due to the unity of the sampling rate that is preset in the algorithm and the sampling interval configured in the simulation environment.

Table 4.5: Statistical outcomes of the experiment (sampling rate = 1/10sec, network scale = 10000 nodes)

Outlier Types	**Number of Original Data Sets**	**Number of Outliers Being Detected Correctly**	**System Average Response Time (Unit)**	**Accuracy**
Error	50	50	3	100%
Exceed threshold	150	150	11.63	100%
Jitter	150	140	9.671	93.3%
Dynamic threshold	150	133	16.68	88.7%

Table 4.6: Statistical outcomes of the experiment (sampling rate = 1/30sec, network scale = 1000 nodes)

Outlier Types	**Number of Original Data Sets**	**Number of Outliers Being Detected Correctly**	**System Average Response Time (Unit)**	**Accuracy**
Error	5	5	3	100%
Exceed threshold	15	15	12.07	100%
Jitter	15	12	9.833	80%
Dynamic threshold	15	12	16.917	80%

4.3 Chapter Summary

In this chapter, we have applied a hierarchical structure for the edge layer in the WSN-Edge-Cloud architecture as presented in the Section 3.1 of Chapter 3 to meet the requirements of rapid response in the intelligent warehouse scenario. The rapid response indicates that by taking advantage of the edge computing, the system gives the first time response to exceptions near the data-generating location rather than waiting for the response strategies from cloud computing, so as to improve the response speed of the system. Instead of the standard approach of an integrated single edge layer in the WSN-Edge-Cloud architecture, this chapter proposes a novel approach of a hierarchical edge layer with cooperated edge nodes in each of the different tiers.

In this hierarchical structure, the edge functions are modularised and abstracted according to the coupling degree of functions and applications. General functions loosely coupled with concrete applications such as data formatting are achieved on low-level edge nodes while the application-specific functions are allocated to high-level edge nodes. Such grading strategy keeps the advantages of edge computing with low latency and enhances the reconfiguration of hierarchical modules at the edge of the network.

Chapter 5

Mobile Agent-based Middleware on WSN Nodes

Wireless Sensor Networks (WSN) applications and infrastructures have proliferated dramatically over the last decade. WSN development challenges are shifting from network deployment to the utilisation of existing resources; from deploying the application-based network to general-purposed networks. This chapter presents a Mobile Agent(MA) cooperation mechanism which can be used by a general-purpose WSN to support the development of demand-based applications. Two types of mobile agents with different functionalities are involved: Patrol Agents and Serve Agents. Applying the proposed cooperation mechanism to mobile agents, the system can achieve efficient cooperation and mutual triggering between the functions of daily patrol, abnormal triggering, abnormal tracking and emergency processing by the mobile agents.

5.1 The Motivation Case for MA Cooperation WSN

As an environment-sensing wireless network system that is gradually becoming an infrastructure nowadays, the WSN is playing a role in different commercial areas, such as smart houses and smart cities, logistics and warehousing, and a variety of industrial scenarios. Some of these application scenarios require strict conditions and have low tolerance, such as dangerous goods storage warehouse.

In hazardous chemicals storage warehouses, dozens of sensitive data need to be monitored in real-time which includes but not limited to volatile gas density, flammable and explosive gas density, real-time temperature, and item shading index. In the traditional warehouse environment, the processing approach of monitoring data is to visualise the original data, which is convenient for the user to monitor at the client terminal. However, when closely depending on human monitoring and judgement, the safety of the warehouse would be tightly related to human errors.

Being motivated under this background, we propose a smart warehouse solution by involving the concept of mobile agents and their cooperation mechanism. Compared with the traditional data processing approach, which tends to upload raw data from thousands of sensor nodes to the centre server while retrieve the processed data to illustrate the visualised information, the mobile agent-based intelligent warehouse system could integrate the functionalities of environment monitoring, abnormal situation alarming, tracking of a dynamically diffused abnormal situation, and first aid control into a whole system without human intervention.

5.2 Requirement Analysis

In an agent-based system, applications are divided into functional separated pieces to facilitate injection and distribution by MA through the network. Functionalities of principal agents and agent execution platform should be clearly analysed to clarify the requirements of designing an agent-based middleware.

An agent is described as a kind of special software that includes unique ID, states, execution codes, as well as the datum. It owns executive resources such as stack, heap, register, and counter. Code execution is supported by accessing the virtual machine. Meanwhile, data can be stored by accessing memory on local nodes.

A backup platform in an agent system is the operating environment of agents. The platform provides a public memory for agents execution; simultaneously, it can allocate and manage those memories. Besides, functions of the virtual machine are achieved on platforms, which include executing a variety of instructions and managing executive resources at runtime. Cooperation among agents is supported by the platform as well. On a single sensor node, multiple agents can be executed concurrently and exchange data through public memories provided by the platform. Agents can be migrated to remote sensor nodes with the same platform. Thus, the management of agents is a necessary function that should be achieved by an agent-based middleware, which should cover but not limited to agent-clone, agent-execute, and agent-kill. Meanwhile, agent message packaging, transmitting, receiving, and debagging are implemented by calling the message interface on the underlying OS.

Apart from the essential functions introduced above, a WSN middleware should also provide network management such as topology management and neighbour list management. The agents should have the ability to acquire network information as well. The middleware should provide the instruction set for agent control and network management for users of this system. The dynamical deployment of agent applications can be implemented by adding an agent generating module and a migration message transceiver.

Therefore, we identify the requirements for the mobile agents and the cooperation mechanism as follows:

Requirements for Mobile agents:

Shared resources: The agents maintain their state, exposed by the sensor nodes. The task state is not tightly-coupled into a physical device. Therefore, the task state is cacheable, and agents provide limited robustness in case of failures.

Independence: The composition of a mobile agent should be platform- and task- independent to address the software and hardware heterogeneity. The agent composition should be adaptive, modified by the hosting devices. The composition can also be exposed as a system resource.

Lightweight composition: Mobile agents must be lightweight in composition, serialisable and transferred as a whole or sequential part.

Requirements for MA cooperation mechanism:

Task-independent: The cooperation mechanism should have a universal standard regardless of the detailed application tasks.

Trigger: Regardless of the order in which the mobile agents are executed, any type of agent can be triggered by another type depending on the requirements.

Human intervention: The cooperative mechanism should have complete closed-loop logic. Except for the initialised program, other routine tasks should avoid human intervention.

Table 5.1: Mobile agent compositions

CoAP message format	**Mobile agent compositions**		**Note**
Header	Header segment	Identifier	To identify the mobile agent
Token		Token	To coordinate requests and responses
Option(s)		Metadata	To describe execute status of task codes
Payload	Body segment	Task codes	To store task codes
Payload		Data	To store target data

5.3 MA Composition and Middleware Platform

Referring to the middleware framework as illustrated in the Section 3.2 of Chapter 3, this section introduces the MA composition and middleware platform on sensor nodes.

5.3.1 Compositions of the mobile agent

As introduced, a mobile agent can be treated as special software with high mobility. In a MA-based WSN system, nodes are the hosts to complete all actions (e.g. data processing and task codes execution) while the objects (data, task codes) compose the mobile agent.

Besides the raw data and task codes, it should contain the metadata to describe data as well as the status of the task codes. Therefore, compositions of a mobile agent are listed in Table 5.1. Adhere to the message format given by the Constrained Application Protocol (CoAP) [56], Table 5.1 shows the compositions of a mobile agent with a header segment and a body segment. The header segment identifies a mobile agent for the nodes, including agent type, metadata, etc. The components correspond to the header, token, and options of the CoAP message. The application specified task codes and the target data are temporarily stored as the payload of the CoAP message, which can be retrieved and processed by the local virtual machine on a node.

5.3.2 Middleware platform support for MA

WSN middleware, which is often developed in the form of a cross-layer architecture [55], is designed to hide complex underlying hardware environments and cumbersome application deployment processes from distributed application developers. The state-of-the-art MA-based WSN middleware [18] [8] usually provides its unique instruction set and MA

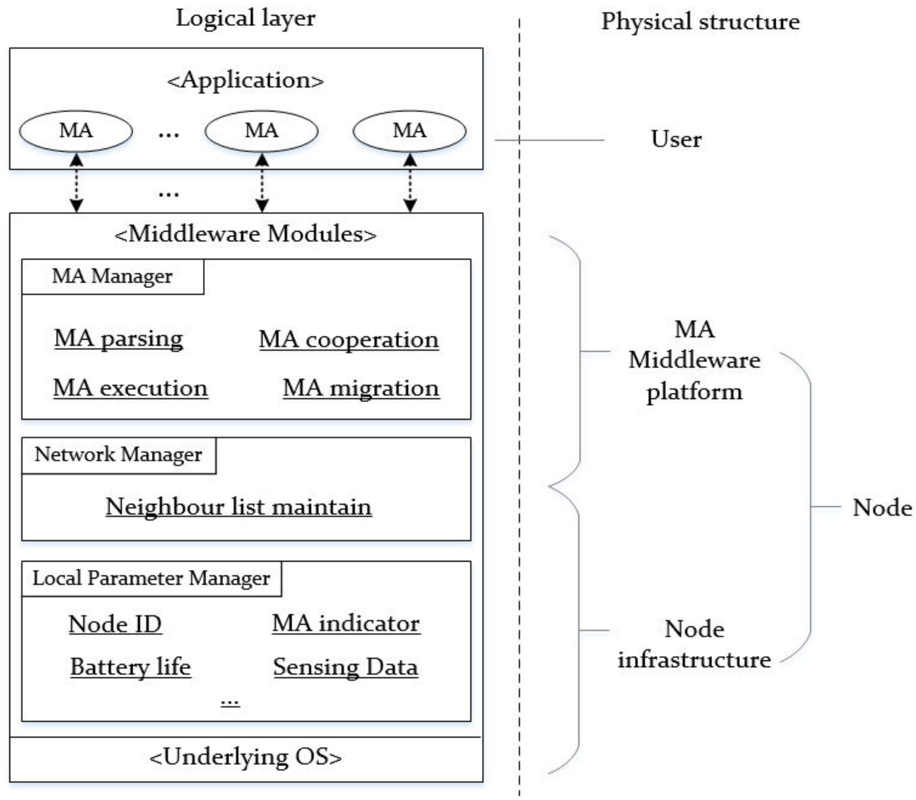


Figure 5.1: Overview architecture of the MA-based system.

interface to the developers by integrating the complex underlying operations and adding the directional service contents.

As shown in Figure 5.1, in addition to the basic network management module and topology management module, the MA management module is also included. These manager modules provide a convenient infrastructure for MA-based applications through cross-layer collaboration. In order to reduce development costs, the middleware platform referred to our work presented in [63] focuses on the CoAP protocol, based on which we can support MA applications that conform to the CoAP message format. At this point, MA-based middleware can provide complete platform support for application development and deployment for real-time needs. In the future development process, compatibility with heterogeneous nodes can also be extended.

5.4 Design and Methodology

Being motivated by the hazardous chemicals storage warehouses scenario as introduced in Chapter 2, we propose the mobile agent's cooperation design and methodology over an MA-based unattended WSN platform in this section. The core of our research is the coordination mechanism between two basic types of mobile agents, which will be refined into three sub-sections.

5.4.1 The overall cooperation mechanism

The target of the cooperation mechanism is to achieve general IoT applications via two universal classes of MAs, which are named 'Patrol Agent' and 'Serve Agent'. The agents carry specialised task codes which could achieve a portion of the application function. The cooperation between two types of agents, therefore, completes overall functions of the application while highlighting the merits of MAs.

The causal conditions for the cooperation mechanism are listed as follows:

(a) The cooperation between MAs depends on the application genre.

(b) Application genre is restricted by the application scenario.

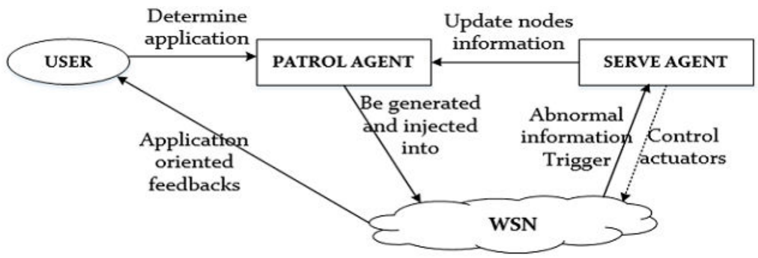
(c) Trigger relationship exists between different types of MAs.

(d) Every type of MA has a unique working mechanism.

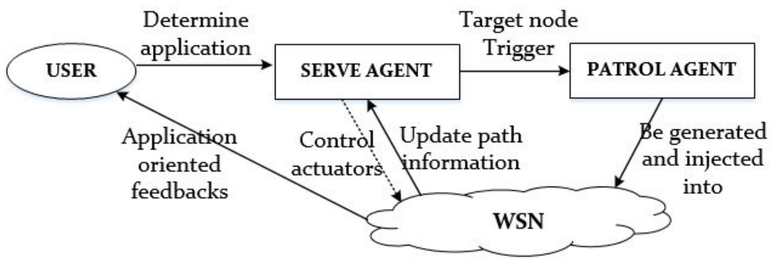
(e) Different patterns of MAs coordination can lead to distinct migration strategies.

The flowchart of the MA cooperation mechanism over a WSN is illustrated in Figure 5.2. To monitor an area in which it may occur the unexpected anomalous sensing data, the pattern that is shown in Figure 5.2 (a) is applied for MAs coordination. After determining the application type, the user generates and injects a Patrol Agent into the network. With the unique working mechanism, the Patrol Agent monitors a specific resource's value on each node. Once an anomaly is noticed, the Patrol Agent will alarm the user and trigger a Serve Agent to further process of the abnormal node. Another pattern of MAs coordination is shown in Figure 5.2 (b). A definite target node is given by the user who injects a Serve Agent with application-specific task codes. Since the route to access the target node is unknown for the Serve Agent, it can trigger a Patrol Agent with destination node ID as the target resource. After determining the path, the Serve Agent can access the target node and serve it as requested.

The distinction between the cooperation patterns shown in Figure 5.2 (a) and (b) is the service provided to the network. The passive service is triggered only by a noticed



(a)



(b)

Figure 5.2: Flowchart of MA cooperation mechanism over a WSN

abnormal node, which indicates the service is possibly not happened in a limited period. The opposite is that when the application has a definite target node, the proactive service is provided to the destination node eventually except for the broken of the network. This implies the inevitability of the service. The detailed unique working mechanism of MAs is introduced in the following sub-section.

5.4.2 Class of mobile agent and working mechanism

The core difference between the two classes of MAs is whether they have determined target before being injected into the network. Through the complementary functions of the two classes of MAs as well as the reasonable cooperation mechanism, the general IoT applications can be realised.

Patrol Agent

The Patrol Agents migrate among physically separated nodes to monitor the value of the target resource. One typical target resource is the sensing data collected by sensor nodes, which is relatively easy to be quantised and compared with a pre-set threshold. Specific metadata stored in the local node could also be regarded as the resource but

Table 5.2: Working mechanism of the Patrol Agent

Working mechanism of the Patrol Agent
Inputs: 1. A set of nodes with periodically updated MAI 2. Random topology 3. A Patrol Agent **Outputs:** 1. ID of the node with target resource 2. A recommended route **Procedure:** 1. Selects an origin node to inject MA 2. For each node been visited by a MA a) Records MA type b) Refreshes local time interval to zero c) Executes MA task code- Compare quantized target resource with threshold; if in the expected range, do 2.d) otherwise, do 3. d) Retrieves MAI of all nodes on the neighbour list, set next node to be the one with the largest time interval 3. For the MA notices unexpected target resource a) Records node ID of the node with unexpected target resource b) Sets origin node ID as the target resource threshold c) Records route nodes d) Concludes task if routed back to the origin node and reports the recorded route

requires more complex processing than the sensing data. Unlike the Serve Agents, Patrol Agents do not have determined route or a definite target node when they are injected into the network.

Hence, one of the patrol strategies for a Patrol Agent is decided by a local parameter Mobile Agent Indicator (MAI) on each node. The MAI on a node records the agent type of the current agent as well as the time interval that the same type of agents visits the current node. The time interval is refreshed to zero at the moment that the same type of agent re-visits the node. With real-time updated MAI on each node in the network, a Patrol Agent always chooses a node with the most prolonged time interval on the neighbour list of the current node. Table 5.2 shows the Patrol Agent's working mechanism following the patrol strategy.

Serve Agent

Comparing with Patrol Agent, the working mechanism of Serve Agents concentrates on providing application-oriented service to a specific node or a set of target nodes. The typical task functions such as “clone” and “match” are mainly achieved by the Serve Agent. As shown in Figure 5.2, the execution of a Serve Agent in both cooperation patterns always leads to an application-oriented stage-feedback. The feedback covers the completed task results or execution states of the uncompleted tasks.

It is worth mentioning that due to the unique features of MAs, whenever a Serve Agent is injected into the network, a Patrol Agent is either triggered to acquire accurate route or has been executed with route feedback already.

5.4.3 Process of MAs on the node

As the host for all the actions on MA processing, the nodes in the network contain the platform to manage and run a MA. The process of a MA arrived at a node is shown in Figure 5.3. Once an MA is received by the node, the MA manager on the node is called to parse the MA type. According to the cooperation mechanism proposed above, the MA manager will activate the patrol strategy for a Patrol Agent migration while collating the annotated resource. This process keeps repeating on all the nodes visited by a Patrol Agent until any anomaly is noticed. The MA manager on the abnormal node will allocate route-back method for the agent along with a Serve Agent trigger information. If the MA manager parses a Serve Agent with the determined route to the target node, it will pass the agent to the target node and directly activate the service method for the agent. Therefore, the task codes that aim to provide service for the target node are executed on the target node. Once the Serve Agent is parsed by the node without any exact route, the node will assign a Patrol Agent trigger information to create a new Patrol Agent. The communication working flow of the MAs among the nodes is shown in Figure 5.4.

5.5 Performance

To test and evaluate the performance of proposed MA migration strategies and cooperation mechanism, we build up the simulation environment and measure the node coverage rate by a single “Patrol Agent” after applying the patrol mechanism.

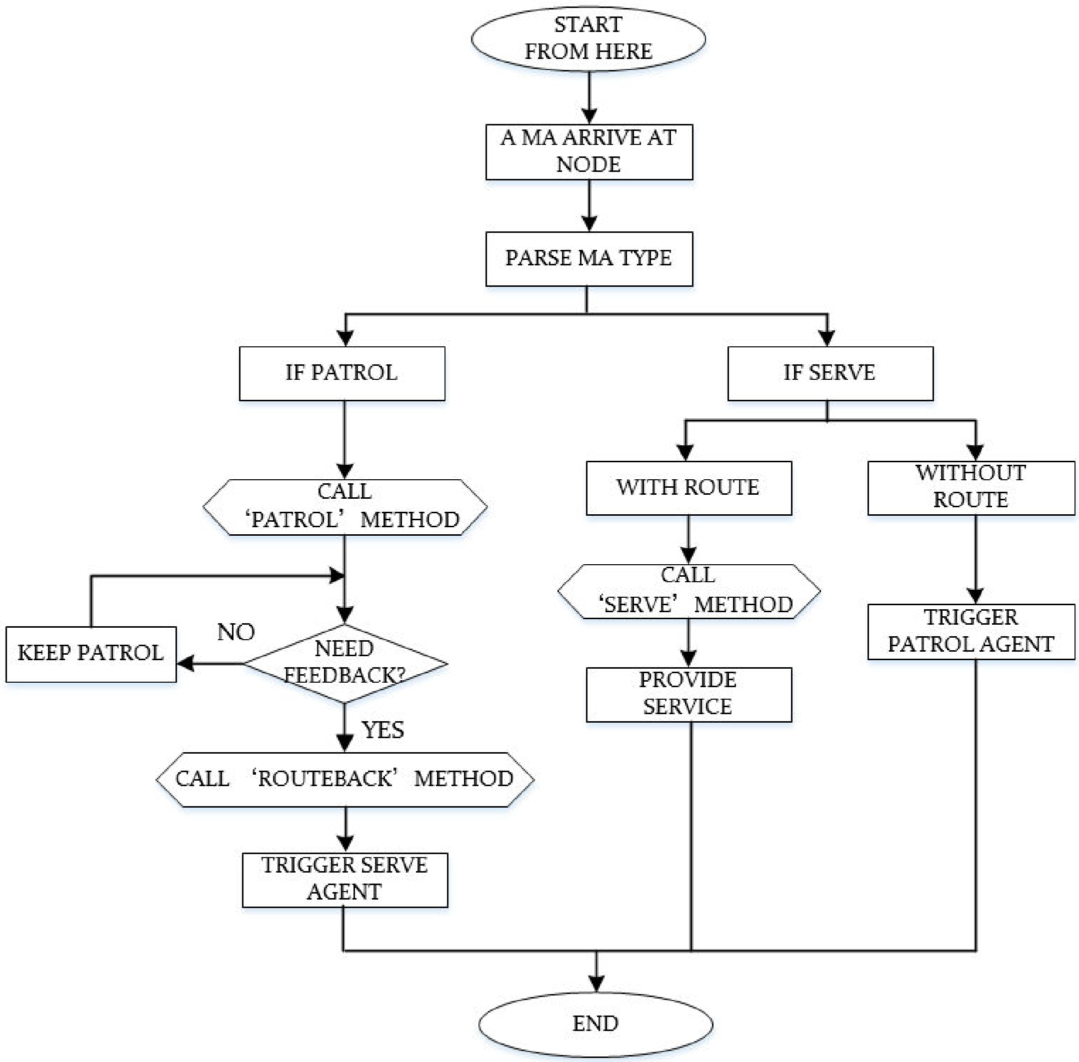


Figure 5.3: Flowchart of the MA processing on the node

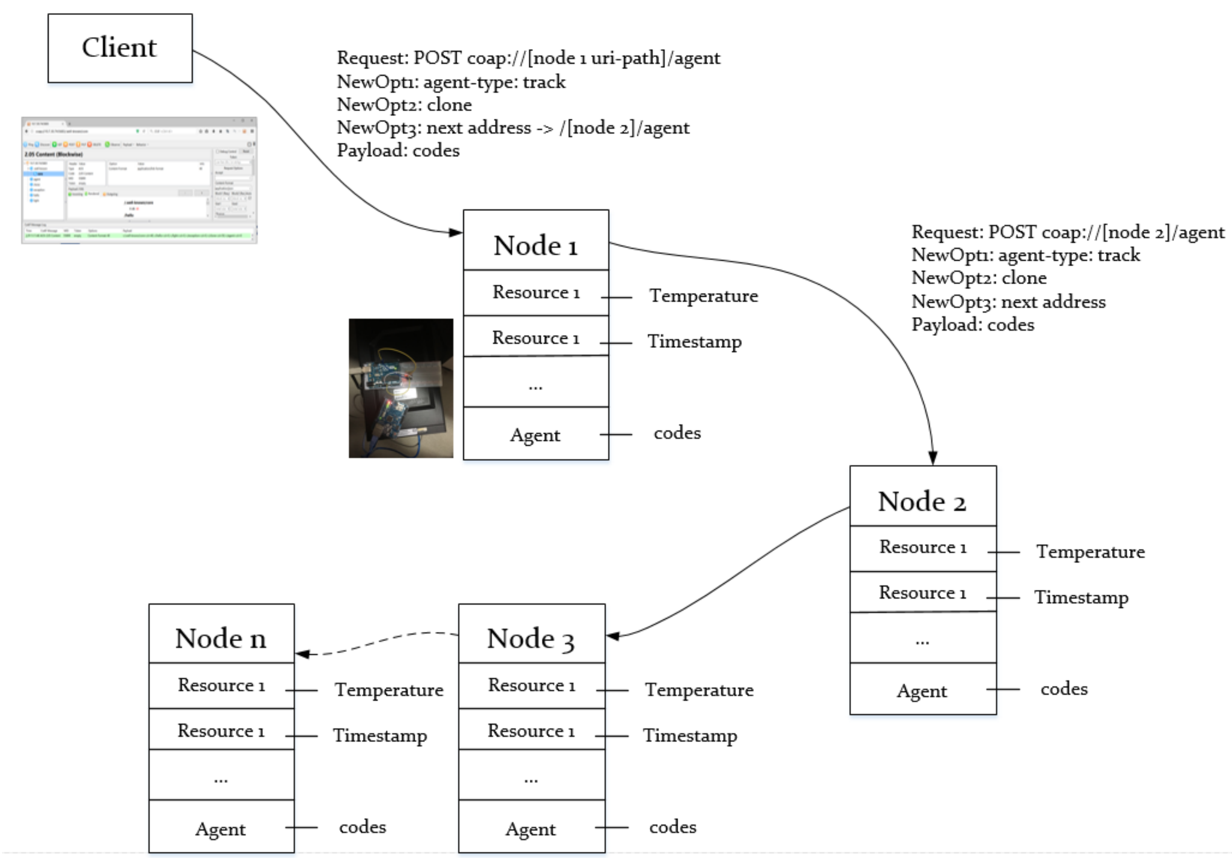


Figure 5.4: Communication working flow of the MAs among the nodes

5.5.1 Pre-set of the simulation environment

In order to simulate the real working environment of a WSN, a topology with randomly allocated nodes is introduced as a precondition of all the tests. In the random network topology, a node will have a variable number of neighbours (parameter *setNeighbour*) which will be an influential aspect for the test results. For the same scale network, *setNeighbour* with enormous value indicates stronger connection among the nodes, which implies that the *setNeighbour* with the same value in the different scale of networks would lead to a closer or more loosely connection among the nodes.

The route length for the MAs over the randomly generated topology is shown in Figure 5.5. The horizontal coordinate indicates the average number of neighbours of each node in the network. By analysing the data distribution and the logarithmic trend with three forward period forecast, it can be observed that as the average number of neighbours increases, the route length for MA gradually stabilises for both scales of the network.

To avoid the default environment of software simulation to be perfect/absolute, a randomly generated "broken nodes pattern" is added into the network, which is used to disable certain numbers of nodes to avoid "perfect condition" during the test. The nodes listed in the "broken nodes pattern" will be removed from the neighbour list of other nodes in the same network. The illustration of disabling nodes from the network is shown in Figure 5.6.

5.5.2 Nodes coverage rate

In the network with randomly selected "broken nodes", the coverage (C) is calculated based on the following Equation:

$$C = (N_{vis} - N_{broken})/(N - N_{broken})$$

, where N_{vis} indicates the number of nodes which have been visited by an MA, N_{broken} denotes the number of broken nodes, and N represents the total number of nodes in the area. To achieve the 100 per cent node coverage rate, the ideal case is that all the nodes except for the "selected broken ones" are visited by a Patrol Agent at least once. It depends on both connections among nodes and patrol strategy.

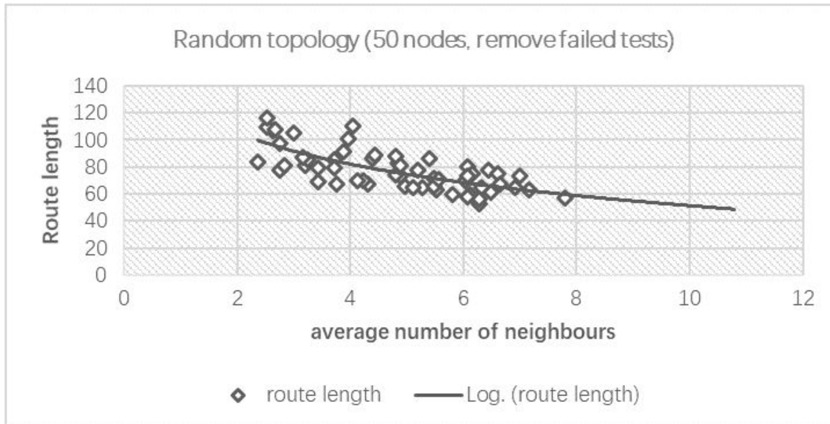
The parameters that may impact testing results are listed in Table 5.3. The number of total nodes indicates the scale of the network. Parameter *setNeighbours* implies the connections among the nodes. The number of broken nodes indicates the performance of

Table 5.3: Parameters setting for coverage testing

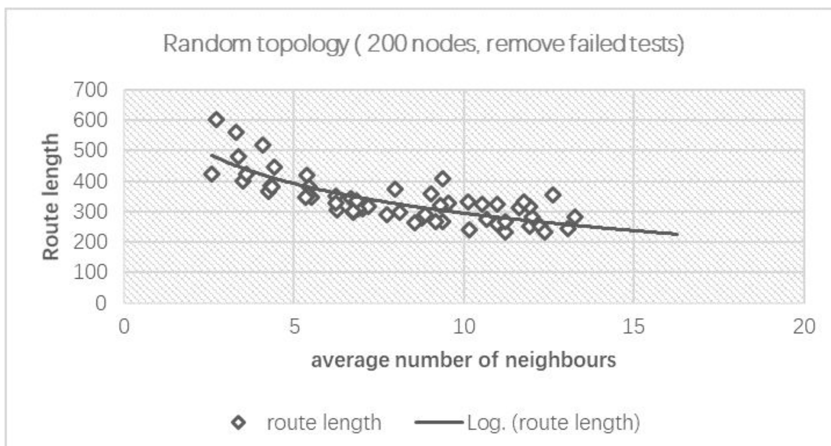
Parameters	**Settings**	**Notes**
total nodes	50, 200, 500	This indicates the scale of the network
setNeighbours (average neighbours)	10 (fixed)	This indicates connections among nodes; with a fixed value, a larger scaled network corresponds to a more loosely connected relation among nodes
brokens	0, 5%, 10%, 15%, 20%, 25%	This indicates the performance of the patrol strategy; with higher percentage of broken nodes, coverage ability of the strategy is stronger at the same coverage level

the patrol strategy. With a higher percentage of broken nodes, the coverage capability of the strategy is stronger at the same coverage level. To avoid unfair influence caused by these parameters, the initial setting of them are listed as well. The testing results are illustrated in Figure 5.7. Since the topology is generated randomly, Figure 5.7 only shows one case that randomly selected from 10 times repeating tests. Regardless of the line smoothness in Figure 5.7, it can be concluded that all the lines have the same trend to be 100% coverage of the nodes. The total number of nodes, which indicates the scale of the network, does not impact the coverage dramatically.

The average value of coverage after repeating test is shown in Figure 5.8. As shown in Figure 5.8, the average coverage of nodes in all the testing network scale is above 98% after ten times repeating tests, which indicates the coverage of nodes applying the designed patrol algorithm under random topology is above 98%.



(a) 50 nodes



(b) 200 nodes

Figure 5.5: Route length for the patrol MA vs. average number of neighbours over different scale of the network

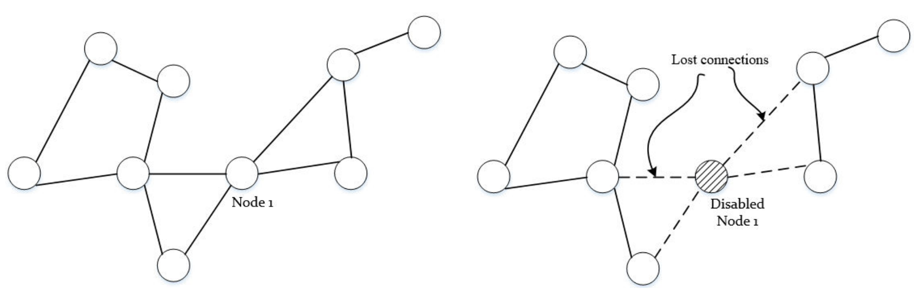
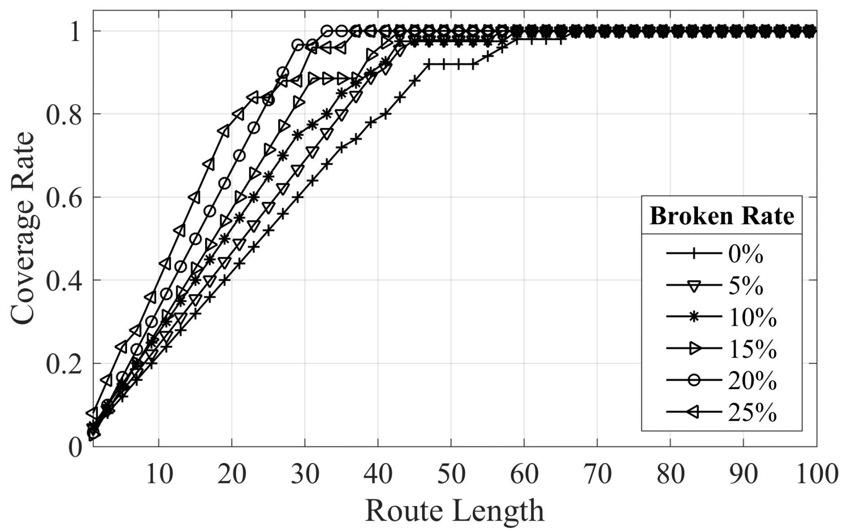
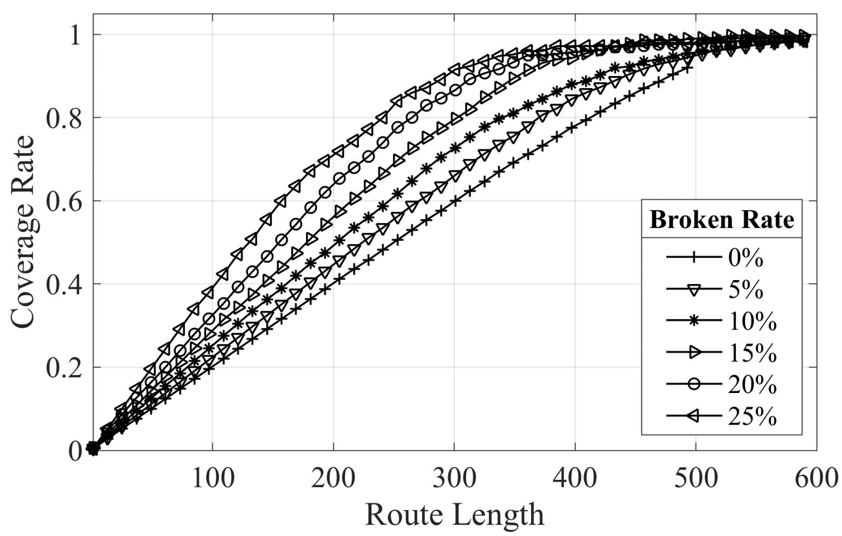


Figure 5.6: An example case after disabling nodes from the network



(a) 50 nodes



(b) 500 nodes

Figure 5.7: Coverage rate over (a) 50 and (b) 500 nodes.

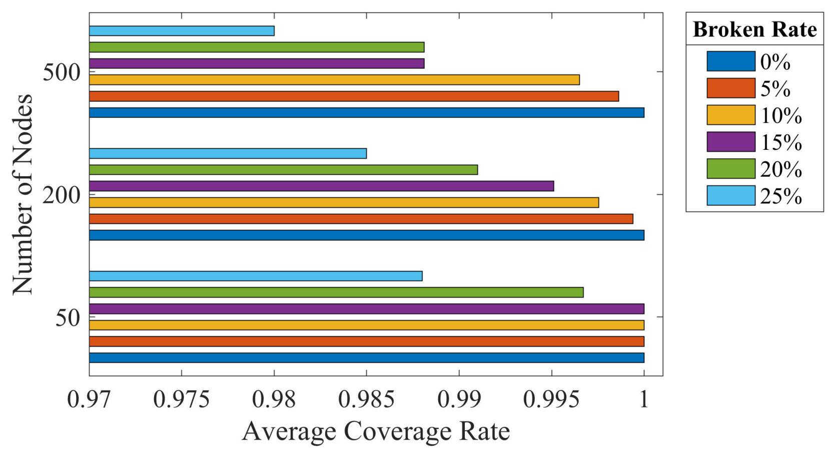


Figure 5.8: Average coverage of nodes over 50, 200, 500 nodes (10 repeating tests).

5.6 Chapter Summary

The chapter presents a mobile agents cooperation mechanism for the general WSN environment. Based on the related research work and being motivated by a practical hazardous chemical storage warehouse scenario, two universal classes of MAs and their cooperation mechanisms are proposed. Through the complementary functions of the MA as well as the suitable coordination mechanism, the general IoT applications can be deployed and executed automatically. The applications achieved by coordinated MAs can respond to environmental mutations in the first place and pre-process these mutations and anomalies without human intervention. Meanwhile, this chpter illustrates the simulation of the proposed MAs based on the randomly generated network topology. Being tested under variety network scale, the node coverage by a MA applied patrol mechanism is above 98%.

Chapter 6

MA-based Middleware on LooCI

As a component infrastructure and event driven middleware system, LooCI is a good option to implement the MA-based middleware framework as introduced in Chapter 5.

6.1 Loosely-coupled Component Infrastructure

The Loosely-coupled Component Infrastructure (LooCI) as presented in [28] is a state-of-the-art and mature middleware system that bridges the gap between distributed WSN applications and diverse platforms. As described in its name, LooCI is composed of a re-configurable component model, a hierarchical system, and a distributed event bus. These features show that LooCI can support a separation of distributed applications from component implementation. The operating system that could support LooCI includes OSGi, Contiki, Squawk, and Android.

Several core definitions in LooCI are listed as follows:

Events: Events are the unit of networked communication and flow between components.

Components: Components are the unit of execution and run in the component runtime environment, which are not interested in where the source and destination of events are.

Codebases: Codebases describe functionality of components, which indicates that each codebase can spawn multiple components.

Event bus: Event bus is an entirely decentralised publish-subscribe communication medium, which can be seen as a distributed implementation of the mediator pattern.

Wires: Wires can be seen as connections between two components.

6.2 Analysis of importing MA in LooCI

As a component infrastructure and event-driven middleware system, LooCI is a good option as the infrastructure to implement MA. The analysis of LooCI operating principle connected with the functional requirements of an agent-based middleware is as follows:

Three core concepts of LooCI are codebase and component, event, and wires. The codebase is an executable code file with a unique node ID that can be deployed to a node while the component is a runtime unit that executes codebase functionality. In other words, codebases provide executable code and components that are the instantiations of codebases, which means all the manager modules (will be presented in coming sections) can be achieved by declaring new codebases and components in LooCI. Components communicate by exchanging events through two interfaces – provided interface and required interface, which are for publishing and subscribing separately. These two interfaces can achieve the message exchange among four modules on supporting platform layer. The exchangable messages include control messages, the status of the agent, neighbour list, etc. An event is a basic unit for network communication. Events are delivered between sensor nodes, which are similar to the role of mobile agents. Every event contains an event ID and payload, which indicates the agent ID as well as storage space for both data and code on agents that could be achieved. Wires are the medium for component communication. Events that have been delivered among diverse components on local or remote sensor nodes which are transmitted through wires.

To clearly show the possibility of implementing an agent-based middleware based on the infrastructure we have proposed, a comparison between LooCI-based agent middleware and typical agent middleware is illustrated in Table 6.1.

As a comparison, several core definitions in a typical agent-based middleware are needed to be specified. Two core components are the agent and tuple space, which have the definitions as follows:

Agent: In classic agent-based system, agents actually belong to the application layer. ‘An Agilla application consists of numerous autonomous agents, possibly of different types, scattered throughout a network’ [13]. This denotes that one application creates a number of agents with interoperated functions to achieve the overall purposes.

Tuple space: The tuple space is local and shared by the agents residing on the node. Special instructions allow agents to access another node’s tuple space remotely.

The left column in Table 6.1 lists the core features of LooCI that are related to MA

Table 6.1: LooCI-based agent middleware and typical agent-based middleware comparison

LooCI-based agent middleware	**Typical agent middleware**
Each codebase can spawn multiple components	Agents can be spawned on a node.
Event manager stores the information of event source and destination in the form of wirings	Have a neighbor list on a node
Standard predefined events should work together regardless of the platform and location. Components which use these standard events should adhere to the format and meaning of the events.	-
LooCI have local wires, outgoing wires and incoming wires. An event matches a wire if all the source attributes the match.	The agent needs to provide a template that matches the tuple to extract a tuple from a tuple space.
	Tuple space provides some operation commands.
Application software is realised in the form of 'compositions' of reusable components.	An Agilla application consists of numerous autonomous agents, possibly of different types, scattered throughout a network.
LooCI allows individual components to be developed on motes at runtime.	Agilla allows agents to be injected on nodes at runtime.
Components are managed by a reconfiguration manager	Agents are managed by an agent manager
Components are uniquely identified by a <node address, component ID>tuple.	A tuple is an ordered set of fields where each field has a type and value.
Wildcards represent no-specific values for the arguments used during reconfiguration and inspection of a LooCI runtime. Their values will never refer to a single, specific instance of their respective argument type.	Templates are unique in that their fields may contain wildcards that match by type
Components, applications and all the other participants interact through events.	Agents migrate carrying their code and state, but no tuple spaces.
In LooCI, using introspection one can discover which components are present on a node along with their interfaces, current state and bindings.	-

middleware system. The right column lists the features of a classic agent-based middleware corresponding to LooCI's features on the left. The grids in Table 6.1 filled by a hyphen mark denote no specific corresponding features.

The comparison shows the possibility of implementing MA on LooCI and the challenge to achieve transplantation in practical since LooCI is event-based which has a gap to a fully functional middleware to support agent system, for instance, the lack of specific tuple spaces.

In simple words, core modules of LooCI can satisfy the functional needs of the MA middleware. The challenge is to declare the manager modules and set suitable properties for the components.

6.3 MA framework on LooCI

Figure 6.1 shows the basic elements in the designed middleware. With the aims of being duplex-friendly to both users and platforms suppliers, the middleware separate core elements into two purposes: one for operations that relate to agents, the other for operations that relate to local components.

Being adapted extensively for the support of native module, LooCI is incorporated and extended by the proposed middleware framework. Individual local components are allowed to deploy on nodes at runtime. As shown in Figure 6.1, components are managed by the reconfiguration manager and exchange events through LooCI event manager. Modules in the "Native" block serve local component reconfiguration while those in "Remote" block serve dynamic deployment of applications. The related working mechanism can be found in [28]. Modules in Remote block extend our research work presented in [62]. Apart from managing agents and related components, tuple space and neighbour list are considered in the overall architecture. Wires of components before and after inserting the agent system into the nodes are shown in Figure 6.2 and Figure 6.3 separately. Before agent system being added into the sensor nodes, each of the components transmit data to sink node directly. Supposing there are three types of sensors on local node, it demands three channels for sensing data transmission. It can be observed that after adding the agent system into a local node, the agent manager will collect sensing data based on application requirement or even aggregate data directly on local nodes before data transmission. Bandwidth requirement will therefore be reduced.

The functionality of the core modules will be presented in the following sections.

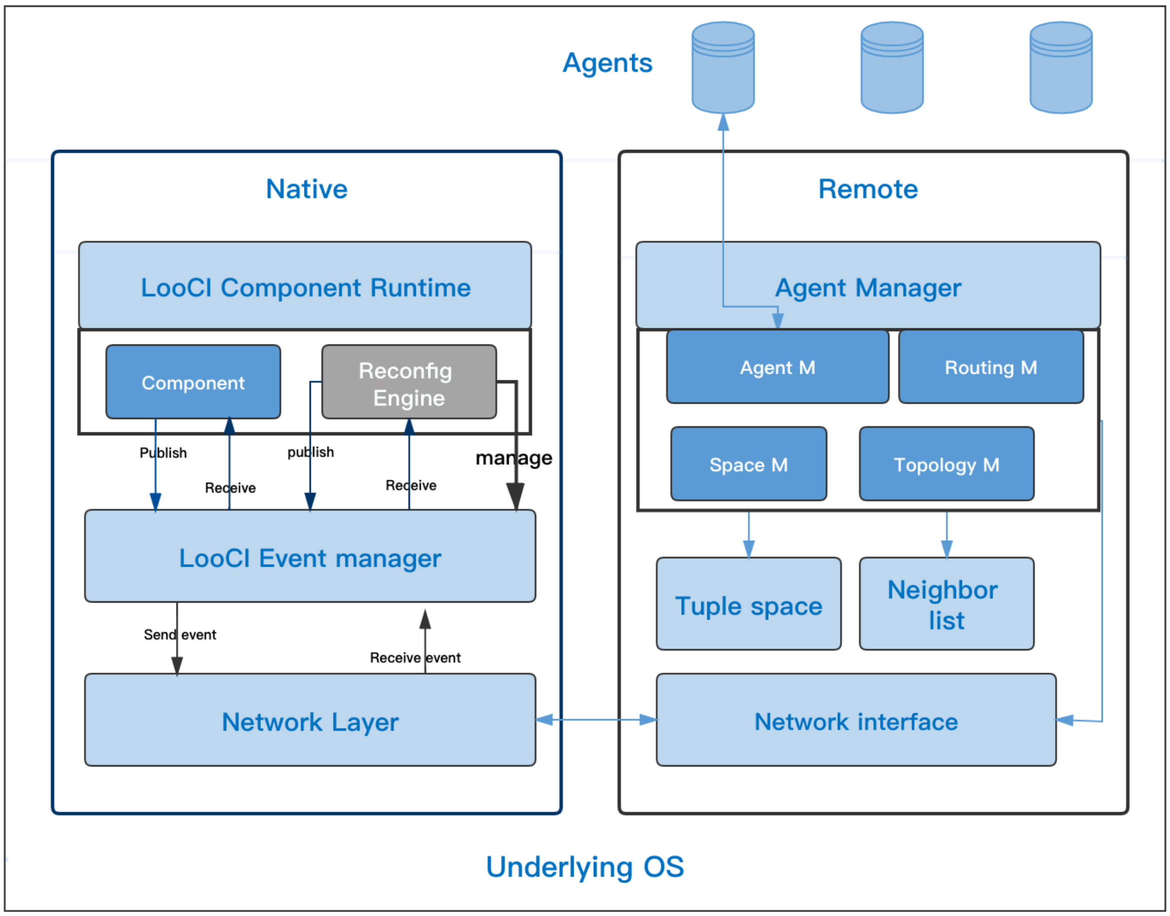


Figure 6.1: Overview architecture design of the middleware.

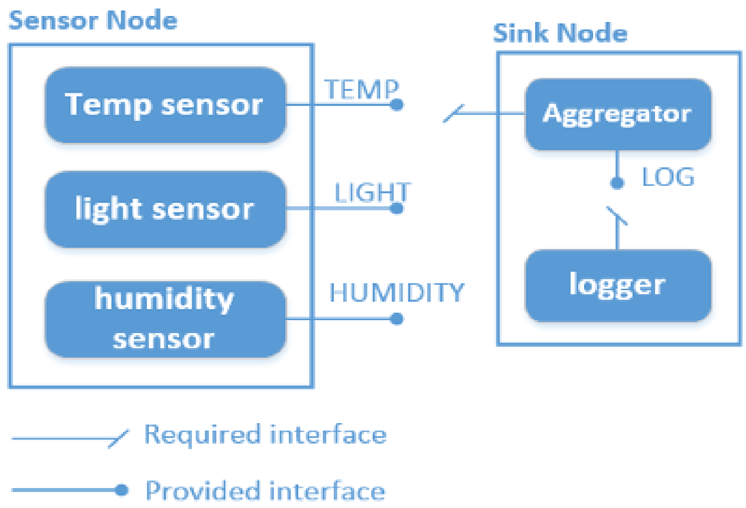


Figure 6.2: Wires of components before adding agent.

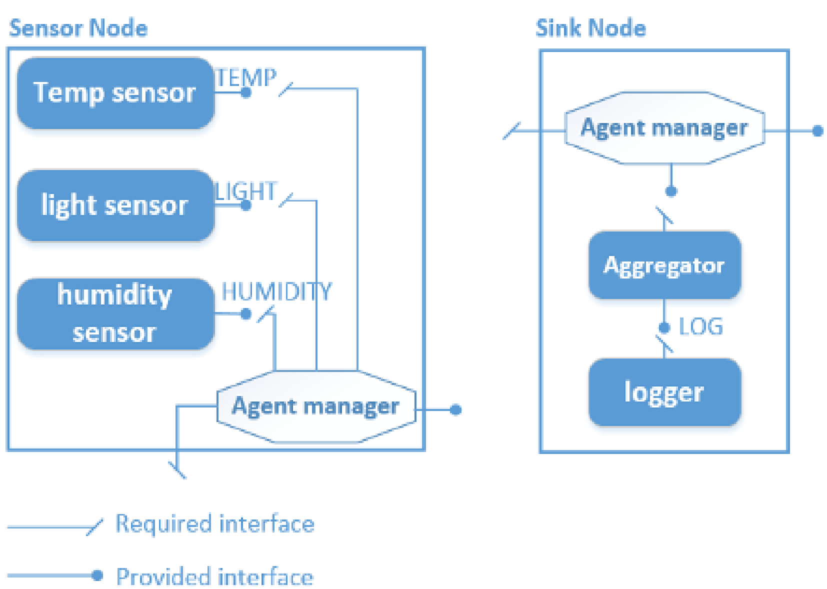


Figure 6.3: Wires of components after adding agent.

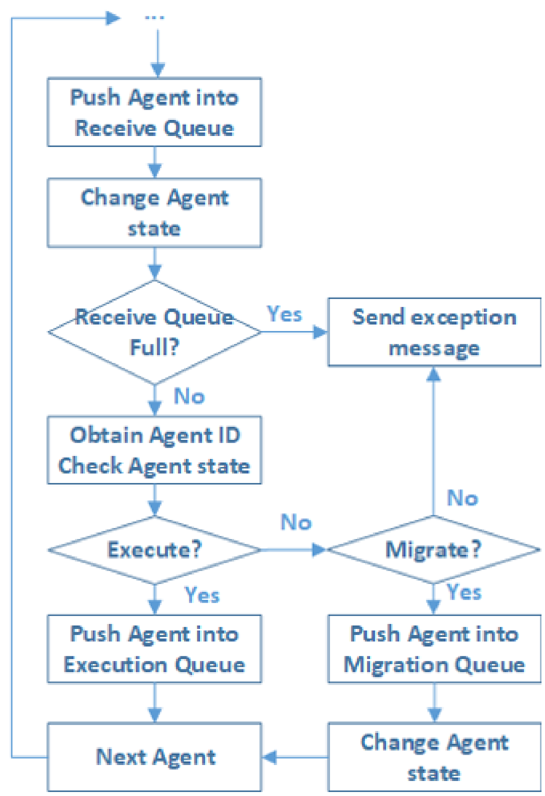


Figure 6.4: Flowchart of an agent's lifecycle in Agent manager

6.3.1 Kernel managers

Block diagrams of four managers in Figure 6.1 refer to our previous research outcomes. In this section, the only core part of agent manager's block diagram is presented for discussion.

In the lifecycle of an agent in agent manager as shown in Figure 6.4, three fundamental functions of agent manager are proposed: (a) switch agent state, (b) allocate resources (frame in queues and memory), and (c) execute agent code.

Figure 6.5 illustrates the basic components and related methods of the agent manager. According to the flowchart of the agent manager, two basic components that are needed to be implemented, which include a monitor and three different functional queues.

Method extract() is one of the core methods of the monitor, which is used to extract the header, payload, purpose, etc. of the events or messages that are received.

pushTo() is another method of the monitor, which is used to push agent body to a

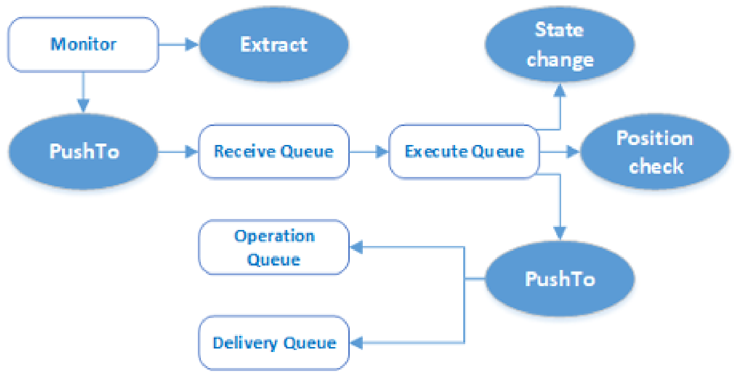


Figure 6.5: Basic components and core methods of the agent manager

suitable queue. This method should be the same for component Monitor and component Queue.

statusChange() is used to change the status of the agents based on their purpose.

positionCheck() is used to check whether there are still empty cells in queues.

6.3.2 Agent transceiver

The mechanism of migration and receiving agents is referred to as agent transceiver in our approach. A simple migration process is separated into three steps: (a) a transmitter judges whether the destination of the current agent is the local node, (b) if it is, the agent arrives destination; if it is not, the transmitter sends the agent to the neighbours through the network, and (c) neighbour nodes will deliver the agent through one-hop routing mechanism until it arrives target node.

During the migration, the transmitter module on the local sensor node coordinates four sub-modules to implement the migration function, as shown in Figure 6.6. Migration modules for agent state, execution codes, instruction stack and others are coordinated together for agent migration. Agent receiver has similar structure to the transmitter. Once the migration process is activated, the coordinator will set up time-up control for delivered parameters and then wait for a response from the target node. Transmission of next parameter will be activated by a correctly receiving expected response within the allotted time, or coordinator will require transmitting failed parameter again. It is the same for the receiver mechanism. Once target node receives the parameter, it will reply a specific message to the source node and then setup time-out control. Memory is re-allocated by

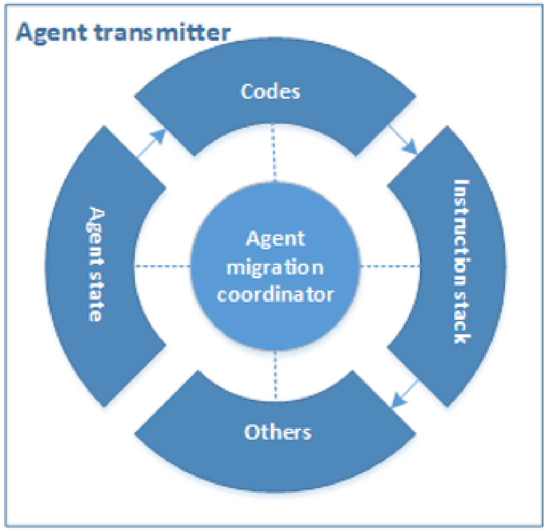


Figure 6.6: Agent transmitter.

Table 6.2: Elements in an agent

ID	Initial node	Target node	Node list	Next hop
State	Register	Instructions	Execution code	Data

agent manager modules in case the time is overdue, and no additional parameters will arrive.

6.3.3 Other modules

A feasible architecture of agents will influence the efficiency of middleware apart from kernel modules in the supporting platform. Table 6.2 illustrates an agent packet, which consists of unique agent ID, initial code, target node, node list, next hop, state, register, instructions, execution code, as well as data. The upper five elements in Table 6.2 are proposed to identify the migration route of the agent, while the bottom five elements are used for agent execution. Necessity of each element in Table 6.2 is presented as follows:

ID: The unique ID identifies which agent is under operation. A unique ID is useful for synchronisation and agent identification for both multiple-agent tasks and single-agent tasks.

Initial node: The initial node identifies where the agent comes from. This element makes it possible to withdraw agents sent by the sink node.

Target node: The nodes that will execute agent codes are set as target nodes. Once an agent arrives on the target node, it switches to the next target node on the node list.

Node list: All the target nodes on agent's route are recorded on the node list. It will be given by the sink node for monitoring purpose.

Next hop: In case that target nodes are not close by (we refer the nodes which can be arrived by agents within one hop as 'close by'), next-hop of the current agent is necessary for continuous migration.

State: Agent state is used to identify the purpose of the current agent: to execute on a local node or is needed to be delivered to other nodes.

Register: The register is used for addressing of next instruction and recording execution status.

Instructions: Instructions are commands to control the activity of agents on the node.

Execution code: The function of applications is achieved through these execution code carried by mobile agents.

Data: After filtering and aggregation of data collected by the local sensors, the agent will carry selected datum back to the sink node.

Adhere to the workflow of agent transceiver and managers as proposed in Figure 6.5 and 6.6, the elements listed in Table 6.2 have already stored in this agent after inserting an agent into sink node by the developers. Each module that is accessed by agents will check and store agent ID. State of the agent is checked and changed by the agent manager. Following the node list which is predetermined, agents on a node will have two operations. For the agents who have already arrived at target node and wait in queue for execution, the virtual machine will be triggered to process instructions in the stack. If the current node is not the destination of the agent, the manager will push this agent into the migration queue waiting for transmission.

6.4 Implementation

Implementation of our proposed middleware is illustrated by two pieces of example codes, and the execution results in the LooCI management console. Table 6.3 shows a part of the codebases in LooCI-Java while Table 6.4 shows a part of the components in LooCI-Java.

Codebase: Besides the component name, provided interface AGENT_EVENT and required interface AGEN_EVENT, STOP_COMPONENT_EV are declared in codebase

Table 6.3: Example code of codebase in LooCI

```
/**
  * LooCIComponent(<agentManager>,
  * <provided interfaces>,
  * <required interfaces>);
*/
public agentManager() {
        super("agentManager",
        //provided interface
        new short[] {EventTypes.AGENT_EVENT };
        // required interface
        new short[] {EventTypes.AGENT_EVENT,
        EventTypes.STOP_COMPONENT_EV } ;
};
```

Table 6.4: Example code of component in LooCI

```
public void receive(short event_id, byte[] payload) {

  /**check eventID to identify

  *control messages and mobile agents

  */

        if(eventID == EventTypes. STOP_COMPONENT_EV){

                System.out.println("Monitored control messages");

                deactivate();

        } else if(eventID == EventTypes.AGENT_EVENT){

                System.out.println("Monitored mobile agent");

                publish(EventTypes.AGENT_EVENT,payload);

        } else {

                System.out.println("received invalid event");

        }

}
```

agentManager(). STOP_COMPONENT_EV is pre-defined in LooCI, which is used as a control message in agent manager. AGENT_EVENT is self-defined to identify agents.

Component: As mobile agents (LooCI event) are deposited to the agent manager, method *receive()* is called to receive agents. In this method, eventID is used to identify control messages and mobile agents. If it is the control message STOP_COMPONENT_EV, the manager will deactivate component. If it is an agent, the manager will publish such an agent. Otherwise, the manager pushes out invalid event information.

To test the implemented components corresponding to the example code shown in Table 6.3 and Table 6.4, we use the command system on LooCI-OSGi. Operations of all components on LooCI console are as shown in Figure 6.7. After "ant" in the directory of agent and agent Manager, it can be checked by command "ls" whether all components are available. Since two components are listed as expected, instruction "deploy" is used for deployment of *agentManager.jar*, and it returns codebase ID 10 by default. After deploying successfully, the instruction "instantiate" is used for instantiating codebase, and it returns component number 10. The deployment and instantiation of the agent are the same as the agent manager except for the returned codebase ID and component number; both are 11 for the agent. After deploying and instantiating, the command "activate" can check whether it is possible to activate all components. The feedback "SUCCESS" appears after "activate" command.

6.5 Chapter Summary

Porting of the mobile agent-based middleware illustrated in Chapter 5 to an event-based component infrastructure - LooCI is proposed in this chapter. Feasibility analysis and requirement analysis are presented as a proof-of-concept. The overview architecture of our design followed with a detailed discussion of specific modules such as kernel managers, agent transceiver, and agent architecture is shown. Examples of codes and implementation are presented by means of a simple test.

Based on the implementation and test results, performance evaluation of our middleware system as well as case studies are necessary to show the applicability of our middleware in the coming future.

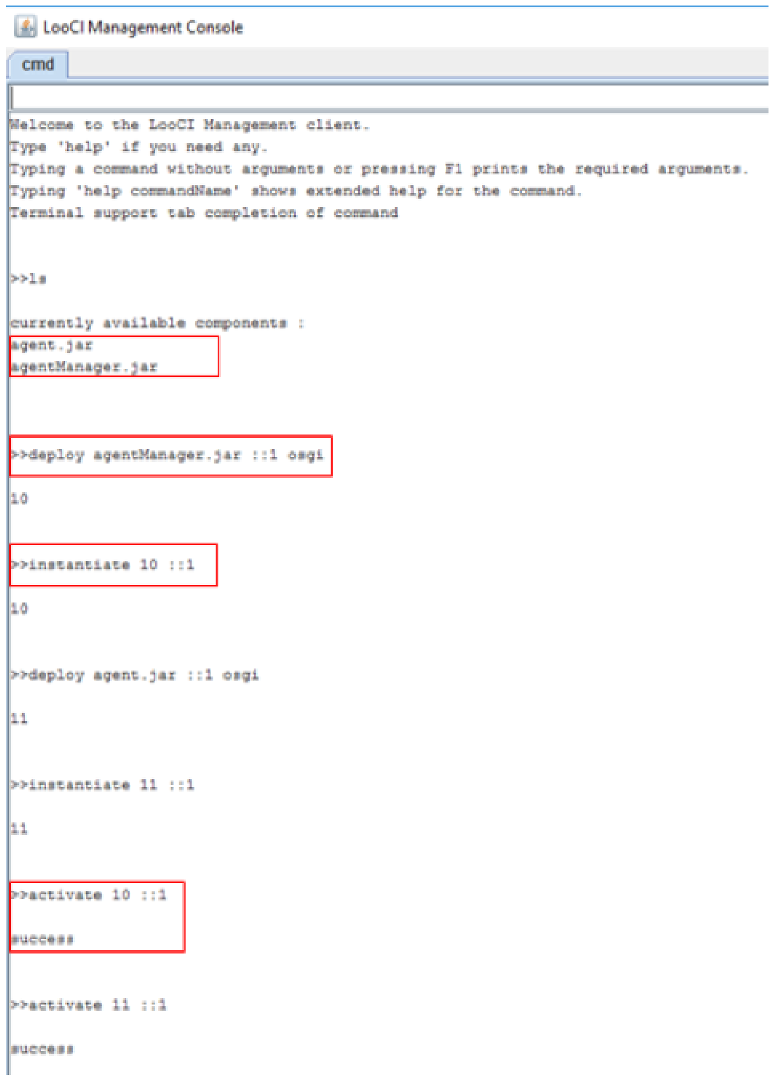


Figure 6.7: Operations of agent manager and agents on LooCI console.

Chapter 7

Conclusions and Future Works

With the popularity of the Internet of Things, an increasing number of applications need the intelligence and interaction of products. As the tentacle of IoT sensing the world, the WSN application system has higher requirements and new challenges for its dynamic, intelligent and real-time performances. Inspired by such requirements and challenges, aiming at how to balance the trade-off between the responding time of a WSN system and the computing complexity of data processing, this dissertation proposes an innovative hierarchical edge computing structure that accompanies a system rapid response strategy, as well as the mobile agent-based middleware framework and multi MA cooperation strategy. With the innovative system architecture and targeted algorithms, the system can support dynamic real-time WSN tasks and ensure the information accuracy by achieving the cooperation between sensor nodes that physically near the data source. Following is the specific outcomes and potential future works.

7.1 Hierarchical Edge Computing

The rapid response system architecture designed for data outlier detection is presented in the thesis, which involves the concept of hierarchical edge computing and brings the advantages of edge computing—its low latency to the edge of the network.

Instead of the standard approach of an integrated single edge layer in the WSN-Edge-Cloud architecture, the research proposes a novel approach of a hierarchical edge layer with cooperated edge nodes in each of the different tiers. In such a hierarchical structure, the edge functions are modularised and abstracted according to the coupling degree of functions and applications. General functions loosely coupled with concrete applications such as data formatting are achieved on low-level edge nodes while the application-specific functions are allocated to high-level edge nodes. Such grading strategy keeps the advantages of edge computing with low latency and enhances the reconfiguration of hierarchical modules at the edge of the network.

For the case study of distributed warehouse management in logistics, an algorithm for distinguishing and rapidly responding to emergency cases have been proposed. Three types of outlier abstracts have been emphasised in the thesis in order to promote the portability of the system. The performance of the system has been evaluated by a software-defined simulation, with a focus on accuracy and rapidness of the reaction by Grade-1 and Grade-2 edge nodes in the system. Applying the rapid response algorithm shows that above 80% of the abnormal cases could be detected and responded to, in less than 20 sampling units.

7.2 Mobile agent-based Middleware

The WSN middleware framework proposed in this thesis takes the mobility of MA as merit into account and highlights the cooperation mechanism between different types of mobile agents. It makes it possible to deploy a general-purpose WSN which could support application deployment at real-time on-demand.

Two universal classes of MAs and their cooperation mechanism are presented. Through the complementary functions of the MA as well as the suitable coordination mechanism, the general IoT applications can be deployed and executed automatically. The applications achieved by coordinated MAs can respond to environmental mutations in the first place and pre-process these mutations and anomalies without human intervention. Meanwhile, the paper illustrates the simulation of the proposed MAs based on the randomly generated network topology. Being tested under variety network scale, the node coverage by an MA applied patrol mechanism is above 98%.

The porting and testing of the middleware on a commercial infrastructure LooCI prove the feasibility of the framework as well as the potential commercial value.

7.3 Future Works

Throughout the case study on the hierarchical edge computing architecture, we have observed that a handful of the jitter cases that were not detected by the system were mainly caused by the ‘break’ during the jitter period. The system judged that the case did not satisfy the predefined jitter condition. One of the potential approaches for improving this is to use a machine learning model for EN-3 so that decisions are made for the lower layers. Importing artificial intelligence models onto Grade-3 edge nodes is one of the avenues to improve the response accuracy, enabling decisions to be made on the edge of the Cloud. The same situation happens to the cases related to the dynamic threshold. Once the deviation of outlier data is not significant, the system may skip the abnormal cases. Thus, to implement the entire system architecture as proposed in this dissertation, a clear direction for future research is the implementation of Grade-3 edge nodes, which potentially focuses on the short-time prediction. Besides the edge computing layer, the interaction and interoperation between the edge and the Cloud are valuable in extending our research. Furthermore, some practical issues in the WSN communication environment such as message collisions, dead nodes, hotspots, etc., are valuable but have not been considered at the current research stage. We consider them as a potential research direction and will do more investigation and experiments to enhance our research outcomes in the future.

During the simulation on the MA-based middleware, the concrete test results of the Serve Agent have not been displayed yet as there is no corresponding comprehensive application. Apart from evaluating the performance of Serve Agents under comprehensive application scenarios, there are also several future work avenues of the research related to the MA-based middleware framework.

To evaluate and deploy the proposed mechanism as well as the whole MA-based system on hardware environment, it is a worthy avenue to implement the framework based on the state-of-the-art protocols. As mentioned, the RESTful CoAP protocol is a relatively mature application layer protocol for IoT scenarios, which has become the preferred protocol template for our implementation. Meanwhile, the message structure and server-client model of CoAP also fit well with the MA working mode. In addition, in order to enhance the generality of our system, we will evaluate our design in different domains besides logistics to remove special features and enhance versatility.

Even though the computing limitation of the WSN nodes has been considered during the research, the power issue and physical energy limitation are weakened. The nodes are powered by chargers instead of a battery. It is valuable to enhance our system by considering the power limitation in the future to expand the application area.

Bibliography

[1] MySQL. Available online: `https://www.mysql.com/` (access on 24 July 2020).

[2] Node-RED. Available online: `https://nodered.org/` (access on 23 June 2020).

[3] PostgreSQL. Available online: `https://www.postgresql.org/` (access on 24 July 2020).

[4] SQLite Home Page. Available online: `https://www.sqlite.org/index.html`(access on 23 June 2020).

[5] D. Afolabi, Ka Lok Man, Hai-Ning Liang, Eng Gee Lim, Zhun Shen, Chi-Un Lei, T. Krilavičius, Yue Yang, Lixin Cheng, V. Hahanov, and I. Yemelyanov. A wsn approach to unmanned aerial surveillance of traffic anomalies: Some challenges and potential solutions. In *East-West Design Test Symposium (EWDTS 2013)*, pages 1–4, 2013.

[6] Basem Y. Alkazemi. Middleware model for TinyOS and contiki-based wireless sensor networks. In *2016 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA)*, pages 1–4. IEEE, 2016.

[7] Rais Amine, Kada Amine, Bouragba Khalid, Ziyati Elhoussaine, and Ouzzif Mohammed. Knowledge discovery in WSN using mobile agents. In *2015 Intelligent Systems and Computer Vision (ISCV)*, pages 1–6. IEEE, 2015.

[8] Faruk Bagci, Julian Wolf, Benjamin Satzger, and Theo Ungerer. UbiMASS ubiquitous mobile agent system for wireless sensor networks. In *2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pages 245–252. IEEE, 2010.

[9] Senthil Murugan Balakrishnan and Arun Kumar Sangaiah. MIFIM—middleware solution for service centric anomaly in future internet models. *Future Generation Computer Systems*, 74:349–365, 2017.

[10] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 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, page 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[11] H. Byun. Decentralized gene regulatory networks: An approach to energy efficient node coordination with delay constraints for wireless sensor systems. *Journal of Communications and Networks*, 21(6):548–557, 2019.

[12] Maurantonio Caprolu, Roberto Di Pietro, Flavio Lombardi, and Simone Raponi. Edge Computing Perspectives: Architectures, Technologies, and Open Security Issues. In *2019 IEEE International Conference on Edge Computing (EDGE)*, pages 116–123, July 2019.

[13] Chien-Liang Fok, G. . Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 653–662, 2005.

[14] Chien-Liang Fok, G. Roman, and Chenyang Lu. Mobile agent middleware for sensor networks: an application case study. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 382–387. IEEE, 2005.

[15] Chien-Liang Fok, G.-C. Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 653–662. IEEE, 2005.

[16] International Electrotechnical Commission. Internet of things: Wireless sensor networks. Available online: `https://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf` (access on 16 July 2020).

[17] International Electrotechnical Commission. Iot 2020: Smart and secure iot platform. Available online: `https://www.iec.ch/resource-centre/iot-2020-smart-and-secure-iot-platform` (access on 9 February 2021).

[18] Lorenzo Corradetti, Daniele Gregori, Stefano Marchesani, Luigi Pomante, Marco Santic, and Walter Tiberti. A renovated mobile agents middleware for WSN porting of agilla to the TinyOS 2.x platform. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–5. IEEE, September 2016.

[19] Sajal K. Das. *Mobile agents in networking and distributed computing. [electronic resource]*. Wiley series in agent technology: 3. Wiley-Interscience, 2012.

[20] Kurt Derr and Milos Manic. Wireless sensor networks—node localization for various industry problems. *IEEE Transactions on Industrial Informatics*, 11(3):752–762, 2015.

[21] Hanan Elazhary. Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. *Journal of Network and Computer Applications*, 128:105–140, February 2019.

[22] Hiroaki Fukuda and Paul Leger. An efficient agent location management for wireless sensor network. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 1014–1019. IEEE, August 2015.

[23] Hemant Ghayvat, Subhas Mukhopadhyay, Xiang Gui, and Nagender Suryadevara. WSN- and IOT-Based Smart Homes and Their Extension to Smart Buildings. *Sensors*, 15(5):10350–10379, May 2015.

[24] W. W. Godfrey, S. S. Jha, and S. B. Nair. On a mobile agent framework for an internet of things. In *2013 International Conference on Communication Systems and Network Technologies*, pages 345–350, April 2013.

[25] R. Godoi Vieira, A. M. da Cunha, L. B. Ruiz, and A. P. de Camargo. On the design of a long range wsn for precision irrigation. *IEEE Sensors Journal*, 18(2):773–780, 2018.

[26] Haosong Gou, M. K. Uddin, and Muhammad Kamal Hossen. An agent-based cooperative communication method in wireless sensor network for port logistics. In *2010 13th*

International Conference on Computer and Information Technology (ICCIT), pages 494–499, 2010.

[27] Cheol-Ho Hong and Blesson Varghese. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms. *ACM Computing Surveys*, 52(5):1–37, September 2019.

[28] D. Hughes, K. Thoelen, J. Maerien, N. Matthys, W. Horré, J. Del Cid, C. Huygens, S. Michiels, and W. Joosen. Looci: The loosely-coupled component infrastructure. In *2012 IEEE 11th International Symposium on Network Computing and Applications*, pages 236–243, 2012.

[29] Danny Hughes, Ka Lok Man, Zhun Shen, and Kyung Ki Kim. A loosely-coupled binding model for wireless sensor networks. In *2012 International SoC Design Conference (ISOCC)*, pages 273–276. IEEE, November 2012.

[30] ISO/IEC. Internet of things (iot)-reference architecture, 2018. Available online: `https://www.iso.org/standard/65695.html` (access on 9 February 2021).

[31] Burak Kantarci and Hussein T. Mouftah. Trustworthy Sensing for Public Safety in Cloud-Centric Internet of Things. *IEEE Internet of Things Journal*, 1(4):360–368, August 2014.

[32] Latif U. Khan, Ibrar Yaqoob, Nguyen H. Tran, S. M. Ahsan Kazmi, Tri Nguyen Dang, and Choong Seon Hong. Edge Computing Enabled Smart Cities: A Comprehensive Survey. *IEEE Internet of Things Journal*, pages 1–1, 2020.

[33] M. N. Khan, H. U. Rahman, M. A. Almaiah, M. Z. Khan, A. Khan, M. Raza, M. Al-Zahrani, O. Almomani, and R. Khan. Improving energy efficiency with content-based adaptive and dynamic scheduling in wireless sensor networks. *IEEE Access*, 8:176495–176520, 2020.

[34] J. Kim, J. In, K. Hur, J. Kim, and D. Eom. An intelligent agent-based routing structure for mobile sinks in WSNs. *IEEE Transactions on Consumer Electronics*, 56(4):2310–2316, 2010.

[35] D. Ko, Y. Kwak, D. Choi, and S. Song. Design of cold chain application framework (ccaf) based on iot and cloud. In *2015 8th International Conference on u- and e-Service, Science and Technology (UNESST)*, pages 11–13, 2015.

[36] R. K. Kodali and N. Sarma. Energy efficient routing protocols for wsn's. In *2013 International Conference on Computer Communication and Informatics*, pages 1–4, 2013.

[37] Ömer Köksal and Bedir Tekinerdogan. Obstacles in data distribution service middleware: A systematic review. *Future Generation Computer Systems*, 68:191–210, 2017.

[38] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, page 8, 2006.

[39] M. T. Lazarescu. Design of a wsn platform for long-term environmental monitoring for iot applications. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 3(1):45–54, 2013.

[40] C. Lei, H. Liang, and K. L. Man. Building a smart laboratory environment at a university via a cyber-physical system. In *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pages 243–247, 2013.

[41] Teemu Leppänen, Jukka Riekki, Meirong Liu, Erkki Harjula, and Timo Ojala. Mobile agents-based smart objects for the internet of things. In *Internet of Things Based on Smart Objects: Technology, Middleware and Applications*, pages 29–48. 2014.

[42] T. Liu, J. Liu, and B. Liu. Design of intelligent warehouse measure and control system based on zigbee wsn. In *2010 IEEE International Conference on Mechatronics and Automation*, pages 888–893, 2010.

[43] X. Liu, N. Xiong, W. Li, and Y. Xie. An optimization scheme of adaptive dynamic energy consumption based on joint network-channel coding in wireless sensor networks. *IEEE Sensors Journal*, 15(9):5158–5168, 2015.

[44] Weidang Lu, Yi Gong, Xin Liu, Jiaying Wu, and Hong Peng. Collaborative Energy and Information Transfer in Green Wireless Sensor Networks for Smart Cities. *IEEE Transactions on Industrial Informatics*, 14(4):1585–1593, April 2018.

[45] Jef Maerien, Sam Michiels, Christophe Huygens, Danny Hughes, and Wouter Joosen. Access control in multi-party wireless sensor networks. In Piet Demeester, Ingrid Moerman, and Andreas Terzis, editors, *Wireless Sensor Networks*, pages 34–49. Springer Berlin Heidelberg, 2013.

[46] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, page 88–97, New York, NY, USA, 2002. Association for Computing Machinery.

[47] Cristian Martin Fernandez, Manuel Diaz Rodriguez, and Bartolome Rubio Munoz. An Edge Computing Architecture in the Internet of Things. In *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, pages 99–102, Singapore, May 2018. IEEE.

[48] Viorel Mihai, Cristian Dragana, Grigore Stamatescu, Dan Popescu, and Loretta Ichim. Wireless Sensor Network Architecture based on Fog Computing. In *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 743–747, Thessaloniki, April 2018. IEEE.

[49] Chen Min, Kwon Taekyoung, and Yong Yuan. Mobile agent based wireless sensor networks. *Journal of Computers*, 1, April 2006.

[50] OASIS. Mqtt version 5.0 oasis standard, 2019. Available online: `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html` (access on 9 February 2021).

[51] Tamoghna Ojha, Sudip Misra, and Narendra Singh Raghuwanshi. Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges. *Computers and Electronics in Agriculture*, 118:66 – 84, 2015.

[52] O. J. Pandey and R. M. Hegde. Low-latency and energy-balanced data transmission over cognitive small world wsn. *IEEE Transactions on Vehicular Technology*, 67(8):7719–7733, 2018.

[53] Jurgo Preden, Jaanus Kaugerand, Erki Suurjaak, Sergei Astapov, Leo Motus, and Raido Pahtma. Data to decision: pushing situational information needs to the edge of the network. In *2015 IEEE International Multi-Disciplinary Conference on Cognitive*

Methods in Situation Awareness and Decision, pages 158–164, Orlando, FL, USA, March 2015. IEEE.

[54] Thomas Rausch, Stefan Nastic, and Schahram Dustdar. EMMA: Distributed QoS-Aware MQTT Middleware for Edge Computing Applications. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 191–197, Orlando, FL, April 2018. IEEE.

[55] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, 3(1):70–95, 2016.

[56] Zach Shelby, Klaus Hartke, and Carsten Bormann. *The Constrained Application Protocol (CoAP)*. Number 7252 in Request for Comments. RFC Editor, June 2014.

[57] Klaas Thoelen. *An Application Platform for Multi-purpose Sensor Systems*. PhD thesis, Ku Leuven, 2016.

[58] Klaas Thoelen, Danny Hughes, Nelson Matthys, Lei Fang, Simon Dobson, Yizhou Qiang, Wei Bai, Ka Lok Man, Sheng-Uei Guan, Davy Preuveneers, Sam Michiels, Christophe Huygens, and Wouter Joosen. A reconfigurable component model with semantic type system for dynamic WSN applications. *Journal of Internet Services and Applications*, 3(3):277–290, 2012.

[59] N. Tziritas, T. Loukopoulos, S. Lalis, S. U. Khan, and C. Xu. Coordination strategies for agent migrations in wireless sensor networks. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 198–206, 2015.

[60] M. Usman, V. Muthukkumarasamy, X. Wu, and S. Khanum. Securing mobile agent based wireless sensor network applications on middleware. In *2012 International Symposium on Communications and Information Technologies (ISCIT)*, pages 707–712, 2012.

[61] V. J. Hodge, S. O'Keefe, M. Weeks, and A. Moulds. Wireless Sensor Networks for Condition Monitoring in the Railway Industry: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1088–1106, June 2015.

[62] Y. Wang, K.L. Man, S. Guan, and D. Hughes. Feasibility analysis of achieving mobile agents for wireless sensor network based on LooCI. In *2017 International MultiCon-*

ference of Engineers and Computer Scientists (IMECS 2017), volume 2228, pages 714–718, 2017.

[63] Y. Wang, K.L. Man, S. Guan, and D. Hughes. A new mobile agent-based middleware system design for wireless sensor network. *IAENG International Journal of Computer Science*, 44(4):591–597, 2017.

[64] Yuechun Wang and Man Ka Lok. Mobile agent framework for WSN based on CoAP. In *Proceedings of the International Symposium on Innovation in Information Technology and Applications*, volume 1, page 1, 2019.

[65] Yuechun Wang and Ka Lok Man. A mobile agent architecture for WSN nodes with REST advantages. *Journal of Industrial Information Technology and Application (JIITA)*, 2(1):7–10, 2018.

[66] Yuechun Wang, Ka Lok Man, Danny Hughes, Steven Guan, and Prudence Wong. A Rapid Response Approach Applying Edge Computing for Distributed Warehouses in WSN. In James J. Park, Laurence T. Yang, Young-Sik Jeong, and Fei Hao, editors, *Advanced Multimedia and Ubiquitous Engineering*, pages 183–189, Singapore, 2020. Springer Singapore.

[67] Yuechun Wang, Ka Lok Man, Kevin Lee, Danny Hughes, Sheng-Uei Guan, and Prudence Wong. Application of wireless sensor network based on hierarchical edge computing structure in rapid response system. *Electronics*, 9(7):1176, July 2020.

[68] Wang S. Wang X., Jiang A. Mobile agent based wireless sensor network for intelligent maintenance. In Huang GB. Huang DS., Zhang XP., editor, *Advances in Intelligent Computing. ICIC. Lecture Notes in Computer Science*, 2005.

[69] L. Xin, L. Jun, Z. Wei, and C. Yuan. Fast and accurate wsn positioning in emergency logistics. In *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 1, pages 277–280, 2010.

[70] Ran Xu, Lili Yang, and Shuang-Hua Yang. Architecture design of internet of things in logistics management for emergency response. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 395–402. IEEE, 2013.

[71] J. Zhan, Y. Sha, and J. Yan. Design and implementation of logistics vehicle monitoring system based on the saas model. In *2012 Fifth International Conference on Business Intelligence and Financial Engineering*, pages 524–526, 2012.

[72] Guangwei Zhang and Ruifan Li. Fog computing architecture-based data acquisition for WSN applications. *China Communications*, 14(11):69–81, November 2017.