DISTRIBUTED INTELLIGENT SYSTEMS FOR CONTROL
WITH DISTRIBUTED HASH TABLE

by

Chi Zhang
M. Sc., Beijing University of Posts and Telecommunications, 2009
B. Eng., Beijing University of Posts and Telecommunications, 2006

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the
School of Engineering Science
Faculty of Applied Sciences

© Chi Zhang 2012

SIMON FRASER UNIVERSITY

Spring, 2012

All rights reserved.
However, in accordance with the Copyright Act of Canada, this work may be
reproduced, without authorization, under the conditions for "Fair Dealing."
Therefore, limited reproduction of this work for the purposes of private study,
research, criticism, review and news reporting is likely to be
in accordance with the law, particularly if cited appropriately.
APPROVAL

Name: Chi Zhang
Degree: Master of Applied Science
Title of Thesis: Distributed Intelligent Systems for Control with Distributed Hash Table

Examine Committee:

Chair: Dr. John Jones
Associate Professor & Acting Director
School of Engineering Science, Simon Fraser University

Dr. William A. Gruver
Senior Co-Supervisor
Professor Emeritus

Dr. Ljiljana Trajkovic
Senior Co-Supervisor
Professor

Dorian Sabaz
Supervisor
CTO, Intelligent Robotics Corporation

Dr. Mehrdad Moallem
Internal Examiner
Associate Professor
School of Engineering Science, Simon Fraser University

Date Defended: December 1, 2011
Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2011
ABSTRACT

As a result of constant demand for the increasing amount of information, traditional centralized computing architectures suffer from lack of robustness, duplication of information, and inadequate system scalability and flexibility. Distributed Intelligent Systems (DIS) show potential in developing management of smart buildings, intelligent transportation, and many other application domains. Peer-to-Peer (P2P) information sharing system, a distributed system that is based on collaboration of peers and efficient use of local computing resources, has the potential to elegantly solve performance issues associated with centralized architectures. In this thesis, we described a prototype based on the Kademlia P2P protocol and a geographic binary tree to illustrate the potential of P2P networks in a distributed intelligent system for control. We implemented a six-node embedded test bed and used it to evaluate the information publishing/querying functionalities. Three sensing peripherals were utilized to demonstrate the flexibility and robustness of the system. These contributions were combined to demonstrate that P2P networks offer a valuable technology for the development of distributed intelligent systems.

Keywords: Distributed Intelligent Systems; Peer-to-Peer Networks; Kademlia protocol.
DEDICATION

To my parents XinQuan Zhang and LanFang Zhou
for their selfless concern and love.

To my supervisors and friends for their kind help and support.
ACKNOWLEDGEMENTS

To Dr. William A. Gruver: Thank you for your kindly suggestions and guidance of all my research. It is my great honor to be one of your students.

To Dorian Sabaz: Thank you for all the meetings and discussions that we had over the past two years. It changed my way of thinking and continues to inspires me.

To Dr. Ljiljana Trajkovic: Thank you for the wonderful lab facilities that you provided over the past years. Thank you for your generous help in my research and thesis writing.
# TABLE OF CONTENTS

Approval .................................................................................................................. ii
Abstract ................................................................................................................... iii
Dedication................................................................................................................ iv
Acknowledgements............................................................................................... v
Table of Contents ................................................................................................. vi
List of Figures ....................................................................................................... ix
List of Tables ........................................................................................................ xi
List of Scripts ....................................................................................................... xii
Glossary ................................................................................................................ xiii

1: Introduction ...................................................................................................... 1
1.1 Evolution of Computing .................................................................................. 1
1.2 Centralized Systems ....................................................................................... 2
1.3 Distributed System ......................................................................................... 3
1.4 Objectives ....................................................................................................... 6
1.5 Contributions .................................................................................................. 7
1.6 Organization ................................................................................................... 8

2: BACKGROUND ............................................................................................... 9
2.1 Introduction .................................................................................................... 9
2.2 Peer-to-Peer Systems .................................................................................... 10
   2.2.1 Unstructured P2P Protocol .................................................................... 12
   2.2.2 Structured P2P Protocol ...................................................................... 13
   2.2.3 Kademlia Protocol ............................................................................... 16
2.3 Comparison between Protocols .................................................................... 19
   2.3.1 Comparison among DHT protocols ...................................................... 20
2.4 Location Awareness ....................................................................................... 21
2.5 Related Research .......................................................................................... 23

3: Systems architecture and Design .................................................................... 24
3.1 Introduction .................................................................................................... 24
3.2 Kademlia Protocol Modification .................................................................. 25
3.3 Geographic Binary Tree ............................................................................... 26
   3.3.1 Basic Design ....................................................................................... 26
   3.3.2 Publishing and Querying ..................................................................... 30
3.3.3 Routing ............................................................................................................. 31
3.4 System Organization .......................................................................................... 32
  3.4.1 Hardware Platform ....................................................................................... 33
  3.4.2 Operating System ......................................................................................... 34
  3.4.3 Network Connection .................................................................................... 35
3.5 System Architecture Design ............................................................................. 36
3.6 Message Packet Design .................................................................................... 38
  3.6.1 HelloRequest Packet and HelloResponse Packet ....................................... 38
  3.6.2 FindNodeRequest Packet and FindNodeResponse Packet ......................... 39
  3.6.3 PublishRequest Packet and PublishResponse Packet ................................. 40
  3.6.4 QueryRequest Packet and QueryResponse Packet ..................................... 41
3.7 Design of Main Functions Sequence .................................................................. 42
  3.7.1 Lookup Process ............................................................................................ 42
  3.7.2 Data Publishing Process ............................................................................... 43
  3.7.3 Data Querying Process ................................................................................. 44

4: System Implementation and ANALYSIS .............................................................. 45
4.1 Introduction ....................................................................................................... 45
4.2 Hardware Selection ............................................................................................ 46
4.3 Software Selection .............................................................................................. 48
4.4 System Development ........................................................................................ 55
  4.4.1 Network Setup ............................................................................................ 55
  4.4.2 User Interfaces ............................................................................................ 56
  4.4.3 Class Explanation ....................................................................................... 58
  4.4.4 Diagrams of the System Domain ................................................................. 62
4.5 System Analysis .................................................................................................. 64
  4.5.1 System Environment Preparation ............................................................... 64
  4.5.2 Experimental Results ................................................................................ 65

5: Open issues and Future Development ................................................................. 86
5.1 Introduction ....................................................................................................... 86
5.2 Open Issues ...................................................................................................... 86
  5.2.1 Ad-hoc Network ......................................................................................... 86
  5.2.2 SQLite Database Performance ..................................................................... 87
  5.2.3 GPIO Utilization ......................................................................................... 88
5.3 Future Development .......................................................................................... 88
  5.3.1 Network Improvement ............................................................................... 88
  5.3.2 P2P Improvement ....................................................................................... 89
  5.3.3 Hardware Improvements ........................................................................... 91
  5.3.4 Data Persistence and Security .................................................................... 92
  5.3.5 Intelligent Autonomous Model .................................................................. 93
  5.3.6 Potential Application .................................................................................. 93
5.4 Final Evaluation and Summary .......................................................................... 93

Appendices ............................................................................................................. 96
Appendix A .............................................................................................................. 96
LIST OF FIGURES

Figure 1.1: Evolution of computing architecture. .......................................................... 1
Figure 2.1: Kademlia node state. .................................................................................. 16
Figure 2.2: Kademlia message routing. .......................................................................... 18
Figure 3.1: Neighbors versus distances. ...................................................................... 28
Figure 3.2: Binary tree based on geographic distance. .................................................. 29
Figure 3.3: Category tree ............................................................................................. 31
Figure 3.4: Routing in a geographic binary tree. ........................................................... 32
Figure 3.5: System architecture. .................................................................................. 37
Figure 3.6: Message transfer. ....................................................................................... 38
Figure 3.7: Hello packets. ........................................................................................... 39
Figure 3.8: Sender information. .................................................................................... 39
Figure 3.9: FindNode packets. ...................................................................................... 40
Figure 3.10: Publish packets. ...................................................................................... 41
Figure 3.11: Query packets. ......................................................................................... 41
Figure 4.1: Development phase system topology. ......................................................... 45
Figure 4.2: Test phase system architecture. ................................................................ 46
Figure 4.3: Experimental node. .................................................................................... 47
Figure 4.4: Command line interface. .......................................................................... 56
Figure 4.5: SWT graphical user interface. .................................................................... 57
Figure 4.6: Android graphical user interface. ............................................................... 57
Figure 4.7: Diagram of the system communication domain. ........................................ 63
Figure 4.8: Experimental system topology. ................................................................. 64
Figure 4.9: Location of test nodes. ............................................................................... 65
Figure 4.10: Binary tree based on XOR distance. .......................................................... 66
Figure 4.11: N2 response with HelloResponse packet. ............................................... 67
Figure 4.12: N5 update routing table. ........................................................................ 67
Figure 4.13: N1 bootstrap procedure. ........................................................................ 68
Figure 4.14: N2 and N3 responses to BOOTSTRAP Request packet. .............. 69
Figure 4.15: Routing table of N1 after bootstrapping. ................................................ 70
Figure 4.16: Movement sequence of the test subject.......................................................... 72
Figure 4.17: N2 publishes first detection to N1 and N4..................................................... 73
Figure 4.18: N3 publishes second detection to N1 and N2.............................................. 74
Figure 4.19: N5 publishes final detection to N2 and N3................................................... 75
Figure 4.20: Query result from N6. .............................................................................. 76
Figure 4.21: Worst case lookup for 16 bit ID................................................................. 78
Figure 4.22: Lookup procedure with additional contact information.............................. 79
Figure 4.23: Lookup messages for different target ID...................................................... 79
Figure 4.24: Linear testing topology.............................................................................. 80
Figure 4.25: Message transfer in original and linear topologies.................................... 81
Figure 4.26: DIN with IP camera.................................................................................. 82
Figure 4.27: Picture’s URL address................................................................................ 82
Figure 4.28: DIN with motion sensor............................................................................ 83
Figure 4.29: Distributed surveillance system............................................................... 84
Figure 5.1: P2P middleware for future development..................................................... 90
Figure 5.2: Future distributed network routing based on DHT........................................ 91
LIST OF TABLES

Table 1.1: Comparison between centralized and distributed systems..........................5
Table 2.1: DHT protocols performance comparison....................................................20
Table 3.1: Location table. ............................................................................................28
Table 3.2: Hardware configurations of BeagleBoard and TS-7300. .........................33
Table 3.3: Comparison of operating systems...............................................................34
Table 4.1: USB WIFI adapter comparison. .................................................................47
Table 4.2: RFID reader comparison. ............................................................................48
Table 4.3: Testbed software environment.....................................................................48
Table 4.4: RFID device system setting. .........................................................................53
Table 4.5: Node querying procedure. ............................................................................76
LIST OF SCRIPTS

Script 4.1: Database SQL scripts ................................................................. 50
Script 4.2: NTP configuration scripts .......................................................... 52
Script 4.3: U-boot code modification ........................................................... 54
Script 4.4: U-boot compiling scripts ........................................................... 54
Script 4.5: GPIO reading scripts ................................................................. 55
Script 4.6: Ad-hoc network setting scripts ................................................ 56
Script 5.1: Command Ping is used to reconnect ........................................... 87
# GLOSSARY

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
</table>
| Ad-Hoc | Ad-Hoc Network  
A network that provides untethered, wireless connection without the help of wired or cellular infrastructure. The main features of an ad-hoc network are: mobile nodes, self-organization to autonomously determine configuration parameters, and system scalability. |
| DHT | Distributed Hash Table  
A hash table that is used to associate data with different peers in a network, where each peer only is aware a subsection of the whole network. |
| DIN | Distributed Intelligent Node  
A hardware platform equipped with an embedded operating system, supporting software, wireless connection and P2P protocol implementation. |
| DIS | Distributed Intelligent System  
A distributed system that can adapt to dynamic environment through peers coordination and entity autonomy |
| EDA | Event-driven Architecture |
| GPIO | General Purpose Input/Output |
| JDK | Java Development Kit |
| JVM | Java Virtual Machine |
| NCP | Network Control Program |
| NIC | Network Interface Controller |
| NTP | Network Time Protocol |
| Overlay | A term to describe that a logical network topology builds on top of a physical topology. |
### P2P
**Peer to Peer network**
A network that is composed with equal privileged peers, where tasks are accomplished through coordination among peers. Peers are resource providers and resource requestors in this network.

### RFID
**Radio-frequency Identification**

### RPC
**Remote Procedure Call**

### RTC
**Real Time Clock**

### SaaS
**Software as a Service**

### SBC
**Single Board Computer**

### SWT
**Standard Widget Toolkit**

### TCK
**Test Compatibility Kit**

### UML
**Unified Modeling Language**
A general-purpose language defined by the Object Management Group to provide a standard, visualized way of complex software design. It is used to describe software architectures and behavior procedures for different functions.

### VTAM
**Virtual Telecommunication Access Method**
1: INTRODUCTION

1.1 Evolution of Computing

Just as evolution in the world of nature, computers and computing architectures are evolving ever since the first computer ENIAC was announced in 1946. The major phases during this evolution are shown in Figure 1.1.

![Figure 1.1: Evolution of computing architecture.](image)

In the first phase of computing, mainframe computing was the only choice due to limited computational resources and the high cost of devices. Furthermore,
only a small group of people had the knowledge and skills to operate mainframe computers. The emergence of personal computers led to a new era of computing where common people began to use the PC as a tool for their business and daily lives. After the Internet ended the era of isolated computers, computers and other smart appliances became a part of this network. Cloud computing, the latest development of computing architectures, is providing users with new web services based on its powerful computational and storage capabilities.

Due to the rapid development of computer hardware, software, and most importantly networking, the world has witnessed an incredible evolution of computing in less than 60 years. With more powerful and cheaper hardware and innovative software engineering, computers are endowed with greater functionalities.

1.2 Centralized Systems

Centralized systems have been the dominant computer architecture since the birth of the computer. This architecture is a compromise between high task requirements and limited computing resources. However, with the explosion of digital information and the appearance of the Internet of Things, whereby millions or even billions of intelligent devices will be connected to the Internet, centralized systems will suffer from the following weaknesses:

1. Information centralization

In centralized systems, the user’s personal information and business records will be stored at remote data centers. Because of the
centralization of data, the user’s privacy is vulnerable to external hostile attacks.

2. Information duplication

Servers have to make a duplication of massive online resources to provide users with useful results. By 2005, Google needed a cluster of 10,000 machines to provide its service but it only searched a subset of available web pages (about $1.3 \times 10^8$) to create its database [1]. With the emergence of massive intelligent sensors and portable devices, data centers will be insufficient to process such large amount of information.

3. System robustness

As the only service providers, the robustness of servers is critical to the system’s well functioning. Failure of the centralized coordinator can potentially cause catastrophic failure of the entire system [2].

1.3 Distributed System

During the past two decades, the weaknesses of centralized systems have forced people to consider a flatter computing architecture. Furthermore, due to the decreasing price of microprocessors, more intelligent devices are gaining the ability to collect, analyze, and exchange information. With millions of information sources, traditional servers will not be powerful enough to cope with them due to the issues mentioned in Section 1.2. While technicians and engineers may have difficulties to cope with structural changes, social insects in
nature (ants, bees, and termites) have already found solutions through use of a distributed systems approach. In fact, a major common point between social insects and current computer networks is how resources are limited, yet the environment is widely changing and unpredictable. Some main features of distributed system are [2]:

1. Instead of finishing tasks in remote servers, the coordination of hundreds of thousands peers is used to achieve the system goal.

2. Coordination is not achieved in a centralized manner. Individual peers do not have a global view of the system but they act upon local information in real time using simple rules and simple responses.

3. The consequence of coordination is not just a collection of small-brained individuals but a complex adaptive system.

Due to these features, distributed systems could help to resolve the weaknesses of centralized systems and provide the following advantages:

1. Decentralization and Hardware Economics

   In centralized systems, service providers have to continually invest in new hardware to satisfy the increasing service needs. While waiting for responses from servers, the majority of client computers are idle. In distributed systems, tasks are accomplished through coordination between peers and content is dispersed among devices. This approach could fully utilize the computational resources of each peer.

2. Scalability and Performance Enhancement
In centralized systems, an increase in the number of clients leads to a decrease in server resources for each client. In distributed systems, with the arrival of new peers, additional computational resources are brought into the system. This feature endows distributed systems with higher scalability and greater capacity to deal with information explosion than centralized systems.

3. Fault Tolerance

Since the single point of failure is removed, the likelihood of distributed system failure decreases with additional peers. Furthermore, each peer could backup its data to other peers. Thus, these peers’ data are preserved with system redundancy.

Table 1.1 describes a detailed comparison of centralized versus distributed systems.

<table>
<thead>
<tr>
<th></th>
<th>Centralized</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing units (PU)</td>
<td>Expensive</td>
<td>Inexpensive</td>
</tr>
<tr>
<td>PU’s information</td>
<td>System wide</td>
<td>Local</td>
</tr>
<tr>
<td>PU’s amount</td>
<td>Small amount</td>
<td>Large amount</td>
</tr>
<tr>
<td>Data collection process</td>
<td>Expensive</td>
<td>Inexpensive</td>
</tr>
<tr>
<td>System robustness</td>
<td>Weak</td>
<td>Strong</td>
</tr>
<tr>
<td>Accomplish tasks</td>
<td>Using commands</td>
<td>Using coordination</td>
</tr>
</tbody>
</table>
Based on the information shown in Table 1.1, the following two elements are keys to a successful distributed system: (1) large numbers of cheap processing units and (2) system coordination mechanism. The first element is addressed in Gorden Moore’s “Moore’s law” proposed in 1965: “The number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years” [3]. Over the past 45 years, computing devices are becoming more powerful, yet their costs are decreasing. Research in peer-to-peer (P2P) protocols and autonomous system models provide a solution with respect to system coordination. Peer-to-peer systems are distributed systems without hierarchical organization or centralized control. In these systems, peers are the basic system blocks and they are equally privileged entities. Peers form self-organizing networks that are overlayed on Internet Protocol (IP) networks to provide service and content to other network peers. P2P systems offer robust wide-area routing architecture, efficient search of data items, selection of nearby peers, redundant storage, massive scalability and fault tolerance [4].

Increasing computing power, decreasing microchips’ price, size, and energy consumption eventually will provide sufficient computing resources that are widely distributed and embedded in the population. Peer-to-peer protocols and autonomous models could provide system coordination and flexibility. Advancements in hardware and software will provide the opportunity for full-scale application of distributed computing.

1.4 Objectives

The objectives of this research are:
• Propose and design a peer-to-peer protocol based system in an embedded environment that resolves the weaknesses of centralized architectures.

• Develop and test a prototype of the system and use it to demonstrate the robustness, flexibility, and scalability of distributed systems.

1.5 Contributions

In distributed systems, the removal of servers provides benefits while also bringing several challenges:

1. Since there are no servers to track information, a mechanism is needed to determine the location of target values, which are the data that requestors need while not knowing their location.

2. In the absence of central commands, each peer in a distributed system should make decisions by itself.

3. Since tasks are accomplished by coordination between peers, there is a need to provide a mechanism for properly transferring messages.

We use the P2P concept to propose a distributed intelligent system for control in an embedded environment. The Kademlia protocol was introduced and modified to fulfill the system requirements. We also designed a protocol to deal with the mismatch between P2P networks and the geographic location information. A system design was given and the proposed application was implemented in Java. We developed a prototype that utilized single board
computers, an embedded operating system with a variety of sensor modalities. This system’s main functionalities were validated and its performance was tested in various scenarios.

1.6 Organization

This Thesis is divided into five chapters. Chapter 2 provides motivation for this research and background information about P2P protocols. Chapter 3 describes the system requirements and the theoretical foundation including system planning, architecture, and design. Chapter 4 presents an embedded system test bed and experimental results. Open issues during the development and experiments, conclusions, and potential future research are summarized in Chapter 5.
2: BACKGROUND

2.1 Introduction

In Chapter 1, we described difficulties that were encountered when utilizing centralized models. As long as this development model is followed, such weaknesses are unavoidable.

In this Section, these difficulties will be re-examined from the point of view of a P2P system paradigm in which agent software is installed in most computers in the network. Each user in this network could choose which part of the resources (storage, CPU processing, network capability) to be shared. Users may start a query and collaborate with other peers in this network to locate their query targets in at most \( \log N \) steps, where \( N \) is the overall number of network peers [4]. This solution is able to cope with the increasing requests on the Internet. Instead of querying from remote servers, users get information from the original information sources directly, thereby ensuring that the information is real-time and always updated. Users can choose the information they are willing to share while protecting their own personal data against external access. By using information sharing instead of information duplication, information explosion in the near future could be accommodated, user privacy is protected, and the mechanism ensures that users always obtain real-time information.
In a distributed computing environment, self-organization is one of the core properties. The ability to realize self-organization, which results in autonomy, is critical to the success of distributed systems.

Four types of autonomous models have been proposed [5]:

1. Systems with dynamically reusable and extensible components. This system could be assigned with new tasks through components extending.

2. Event-driven architecture (EDA) and context-aware systems designed to sense events, filter them, and to decide subsequent actions.

3. Goal-based and environment model-based intelligent systems that dynamically plan their own actions to achieve a system goal.

4. Pre-configured systems with in-built local goals to define system execution and self-regulated without global control.

We selected the EDA model for our research because it is easy to implement and events may be represented in the form of system messages.

2.2 Peer-to-Peer Systems

The peer-to-peer (P2P) system paradigm is not new. During the 1990s, there were already several investigations of this architecture [6], [7]. However, due to hardware, software, and network limitations, auxiliary equipment and programs such as the Virtual Telecommunication Access Method (VTAM) and Network Control Program (NCP) were widely used to ensure the distributed connections. In Simon’s system [6], core function message routing was
accomplished by predefining all nodes in each host’s VTAM and the connected NCP. This approach showed the feasibility of P2P systems but did not show the potential power of the P2P architecture.

Ten years later, P2P network research evolved into a period of rapid development. Several P2P protocols were designed and reported, including Pastry [8], Tapestry [9], Chord [10], CAN [11], Kademlia [12], in addition to commercial P2P applications such as Napster and Gnutella. During that period, a formal definition of a P2P network was given by Rudiger who believed that the most distinctive difference between centralized and P2P networks was the concept of a Servent, derived from server (“Serv-”) and client (“-ent”). Thus, a Servent represents the capability of a P2P network to act as a server and a client at the same time (as Janus in the ancient Roman religion who had two faces on his head).

The following definition that reveals the essence of P2P network:

“A Peer-to-Peer (P-to-P, P2P) network is a distributed network architecture where participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, etc.) for providing service and content to other network participants (e.g., file sharing or shared workspaces for collaboration). The participants of such a network are thus resource providers as well as resource requestors (Servent-concept) [13].

Rudiger described the following two types of P2P networks:
• Pure P2P: A network that is a P2P network according to the definition and where if any arbitrary chosen entity can be removed from the network without network suffering loss of network service.

• Hybrid P2P: A network that is classified as a P2P network according to the definition and where a central entity is necessary to provide parts of the offered network services.

The first challenge of our research is lookup, one core function of P2P system that is utilized to determine the location of target values. In order to realize this functionality, two mechanisms were designed [4]:

• Unstructured mechanism: An unstructured P2P network composed of peers joining and leaving the network with some loose rules. These peers do not have prior knowledge of the system topology. This type of network depends on message flooding to locate the target value.

• Structured mechanism: The structured P2P network topology is tightly controlled and contents are assigned with specified locations that will make subsequent queries more efficient. Such structured P2P systems mainly utilize the Distributed Hash Table (DHT) as the basic data structure, where a value's location is determined by a distance computation between node identifiers and the target value's unique key.

2.2.1 Unstructured P2P Protocol

1. Freenet:
Freenet is an adaptive P2P network of peers that stores and retrieves data items through querying, where these items are identified by location-independent keys. Each peer maintains a dynamic routing table that contains addresses of other peers and the data keys that they are holding. In Freenet, requests for targets are passed from peer to peer through a chain of proxy requests in which each peer makes a local decision about the next location to send. Freenet enables users to share unused disk space [14]. This system is not intended to guarantee permanent file storage.

2. Gnutella:

In the Gnutella protocol (version 0.4), when the user wants to search for a target value, the Gnutella client sends the request to all its actively connected nodes and these nodes in turn forwarded the requests to their connected nodes. This search procedure ends when it finds the target or reaches a predetermined number of hops from the sender (maximum 7 hops) [15]. Such protocol design is extremely resilient to peers entering and leaving the system. However, the current search mechanisms are not scalable and generate unexpected loads on the network [4].

2.2.2 Structured P2P Protocol

1. Content Addressable Network:

CAN utilizes a $d$-dimensional Cartesian coordinate space to generate a virtual logical address. This logical address is independent of the physical location of network peers. Each peer in this system is responsible for a virtual coordinate zone and this peer is identified by the boundaries of this zone. A key
is mapped onto a point in this coordinate space and it is stored at the peer that is in charge of the zone where the key resides. Each peer maintains a routing table of all its neighbors in coordinate space. Two peers are neighbors if their zones share a d-1 dimensional hyperplane. The look-up operation is implemented by forwarding the query message to the neighbor that is closest to the destination [4], [11].

2. Chord:

Both keys and peers in Chord are assigned a unique m-bit ID from the same one-dimensional ID space. A logical ring is formed among all the peers in the network. This ring owns positions from 0 to $2^m$-1, where ID 0 follows the highest ID. Key $k$ is saved in its successor peer, which is defined as the peer whose ID most closely follows $k$. Chord performs lookups in $O(\log N)$ time, where $N$ is the number of peers. Each Chord peer maintains a finger table that contains the IP address of the peer that is halfway around the ID space from the original peer, a quarter-of-the-way, and so forth in powers of two. A peer forwards a search for key $k$ to the peer in its finger table with the highest ID less than $k$ [4, 10].

3. Tapestry:

Tapestry utilizes the idea of Plaxton mesh [16]. It maps peer and key IDs into strings of numbers. In a given level of a peer’s neighbor map, a number of peers that match up to a certain position of this peer’s ID are included. The $i$th entry in the $j$th level is the ID and the location of the closest peer that ends in “$/i$+suffix(N, $j$-1).
To route the message, the (n+1)th level map will be checked and the entry that matches the value of the next digit in the destination ID will be looked up. Assuming consistent neighbor maps, this routing method guarantees that any existing unique peer in the system will be obtained within at most log\(N\) logical hops, where \(N\) is the size of the system [9].

The original Tapestry data structure, which works well in a static environment, is unable to support dynamic joining and leaving of peers. Later versions added support for such dynamic operations, but the emphasis on proximity made them complex [17].

4. Pastry:

In the Pastry protocol, each node is assigned with a 128-bit node ID and these IDs are assumed to be evenly distributed. In the Pastry network with \(N\) nodes, a given key could be located in less than log\(N\) steps.

In each search step, the Pastry node forwards the message to the next node whose node ID shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key already shares with the present node ID. If no such node is found, this message will be forwarded to a node whose node ID shares the same length prefix with the current node but is numerically closer to the key than the current node ID.

Each Pastry node maintains a routing table, a neighborhood set, and a leaf set. The routing table and leaf set are utilized to route messages and the neighborhood set is useful for maintaining the node’s locality properties [8].
2.2.3 Kademlia Protocol

Another structured P2P protocol based on DHT is Kademlia, designed by Petar Maymounkov and David Mazieres in 2002. In this protocol, each target value and node are assigned a 160 bit ID. To determine the distance between network nodes, the XOR metric is introduced (XOR distance between node 1110 and node 1111 is 0001 in binary format) [12].

2.2.3.1 Node State

Since each node ID has 160 bits, the entire node space of Kademlia is $2^{160}$. In this protocol, a node state indicates how a node keeps information about other network nodes. Figure 2.1 shows the partition of Kademlia sub-trees for a single node in this network:

![Figure 2.1: Kademlia node state.](image)

In Kademlia, a node’s sub-tree is defined as follows:

Kademlia nodes store contact information of their “neighbors” for messages routing. For each $0 \leq i < 160$, there exists a subset of node space whose node XOR distance to the initial node is between $2^i$ and $2^{i+1}$. These lists are sub-trees...
of one node. By selecting \( k \) nodes from these sub-trees, \( k \)-buckets are created for this initial node.

For node 0011 shown in Figure 2.1, if \( i \) equals 0, \( k \) nodes from sub-tree 0 will be selected and put into \( i \)th \( k \)-bucket. Also, if \( i \) equals 1, 2, 3, sub-tree 1, 2, 3 will be selected. In the end, node 0011 will hold partial contact information about any sub-trees in the entire node space.

For small values of \( i \), the \( k \)-buckets will generally be empty (since no suitable nodes will exist). For large values of \( i \), the lists may grow up to size \( k \), where \( k \) is a system-wide parameter. \( K \) is chosen to ensure that any given \( k \) nodes are unlikely to fail at the same time within an hour (in practical applications, \( k \) is set to 20) [12].

### 2.2.3.2 Message Routing

In this Section, we describe the basic approach for Kademlia’s message routing. The line segment at the top of Figure 2.2 represents the node space of 160-bit IDs and it indicates how the lookup actions converge to the target node. Node 0011’s first lookup destination is node 101 because node 101 is in node 0011’s neighbor list. All the following lookup steps are based on the information that is returned by the previous Remote Procedure Call (RPC), an inter-process communication that allows a computer program to execute a procedure in another address space [31]. As shown in Figure 2.2, after four lookups, node 0011 reaches its target node 1110.
Figure 2.2: Kademlia message routing.

Four RPCs are defined: PING, STORE, FIND_NODE, and FIND_VALUE [12].

PING RPC’s main function is to work as a probe to check if one node is still online. STORE RPC ensures that other nodes could store a <key, value> pair for future information querying. FIND_NODE RPC takes a 160-bit ID as an argument. The recipient of this RPC should return a list of <IP address, UDP port, Node ID> triples which are the nodes the recipient knows close to the target ID. FIND_VALUE RPC’s mechanism is similar to FIND_NODE. Instead of returning a list of results, if one RPC recipient owns the key value that is identical to the querying argument, it will return the stored value and the entire procedure immediately terminates.

The core function of Kademlia, as with all other DHT implementations, is to locate the $k$ closest nodes to the querying target ID (the lookup function). The
lookup initiator starts by selecting $\alpha$ nodes from its closest non-empty $k$-bucket from local routing table, where $\alpha$ is a system-wide concurrency parameter. The initiator then sends parallel, asynchronous FIND_NODE RPCs to the $\alpha$ nodes it has chosen. In the recursive step, the initiator resends the FIND_NODE RPCs to nodes it has discovered from previous RPCs. From the $k$ nodes that the initiator has discovered closest to the target, it selects $\alpha$ nodes that it has not yet queried and resends to them the FIND_NODE RPCs. The lookup terminates when the initiator has queried and obtained responses from the $k$ closest nodes it has seen [12].

Most Kademlia operations are implemented based on the lookup procedure. To store a <key, value> pair, a participant locates the $k$ nodes that are close to the key and sends STORE RPCs to these nodes. To find the <key, value>, a node starts by performing a lookup to find the $k$ nodes with IDs closest to the key.

2.3 Comparison between Protocols

Although unstructured protocols are easy to realize, their flooding mechanism for querying may cause excessive network traffic [4], an issue that affects scalability. Even though structured protocols may have to add extra information to each packet they are transferring, this information will be utilized for determining the target’s location. The method provides a lookup within $\log N$ steps, which enables effective scalability of the system. Therefore, a structured protocol is preferred as the system protocol.
2.3.1 Comparison among DHT protocols

The following table gives a detailed comparison between structured P2P protocols that are based on DHT.

Table 2.1: DHT protocols performance comparison.

<table>
<thead>
<tr>
<th></th>
<th>CAN</th>
<th>Chord</th>
<th>Tapesstry</th>
<th>Pastry</th>
<th>Kademlia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node state</td>
<td>$d$</td>
<td>log$N$</td>
<td>log$N$</td>
<td>log$N$</td>
<td>log$N$</td>
</tr>
<tr>
<td>Lookup</td>
<td>$dN^{1/d}$</td>
<td>log$N$</td>
<td>log$N$</td>
<td>log$N$</td>
<td>log$N$</td>
</tr>
<tr>
<td>Peers join/leave</td>
<td>$dN^{1/d} + d\log N$</td>
<td>$(\log N)^2$</td>
<td>log$N$</td>
<td>log$N$</td>
<td>log$N + c$</td>
</tr>
<tr>
<td>Routing performance</td>
<td>$O(dN^{1/d})$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N) + c$</td>
</tr>
</tbody>
</table>

Node state indicates the number of other nodes that each node knows about. Lookup indicates how many messages (Internet packets) are required for each operation. $N$ is the total number of nodes in the system, $d$ is CAN's number-of-dimensions parameter, and $c$ is constant for Kademlia.

Due to CAN’s hyperspace design, the cost of its lookup method grows faster than for other protocols [17]. Thus, it is not as effective as other DHT protocols. Since Tapestry and Pastry are based on the Plaxton Mesh data structure, their performance is not effective when dealing with dynamic network topologies [16]. Chord’s unidirectional feature causes dramatic performance degradation in the presence of hostile node attacks [30].

Kademlia has the following advantages compared with other DHT protocols [12]:

20
• In contrast to a ring or hyperspace structure, tree-based routing could speed up the lookup procedure.

• It decreases the number of configuration messages that nodes should send to learn about each other. Configuration information spreads automatically as a side effect of key lookups.

• To speed up the lookup procedure, Kademlia uses parallel, asynchronous queries to avoid timeout delays from failed nodes.

• XOR metric's symmetric and transitive features ensure that Kademlia nodes could receive lookup queries from the same distribution of nodes contained in their routing tables.

• Kademlia uses a single routing algorithm from start to finish, ensuring the simplicity of protocol implementation.

Based on these considerations and detailed understanding of protocol based on numerous reports [12], [24] - [27] and extensive code resources [28], [29], we have selected Kademlia for our P2P protocol realization.

2.4 Location Awareness

If a device knows its location, it can adapt its behavior in significant ways without requiring any artificial intelligence [18].

One important feature of Kademlia is the utilization of the XOR metric for distance measurement between nodes in the node space. This operation greatly simplifies the procedure of node distance calculation and peer lookup. One drawback of this operation is that the Kademlia network structure cannot be
readily mapped to the real-world topology [32]. One node could publish its information to its XOR neighbors. However, in a real-world topology, these neighbors may be hundreds or thousands miles away, implying that the transmission traffic load could increase due to the poor selection of nodes.

Let us consider the following scenarios: Local store manager Bob would like to distribute advertisements for his small store. He may purchase Google Search Keywords to publish his advertisements online or he could print some brochures for delivery to his neighborhood. Jamie has moved from another place and has no knowledge of her new town. In this situation, a local person who is familiar with this location is more useful than a "powerful" search engine.

The above scenarios illustrate the two approaches that are available with respect to information publishing or querying, regardless of the role being played by the user:

- Keywords oriented publishing/querying where keywords are used to determine a list of relevant search results. This information should be openly published (websites, online yellow pages) in order to be indexed by search engines.

- Location oriented publishing/querying where location is used to filter out useless activities/information. In particular, information could be published in a subset of the network instead of the overall network. Regarding the information querying, relevant nearby information should be more useful than information that is hundreds of miles away.
Since most recent P2P networks utilize a DHT based on the hash value of a data object, keyword publishing/querying is the main solution being used. To cope with the mismatch between DHT and geographic locations, we introduce location awareness into P2P networks.

In Chapter 3, we design a P2P network that contains both a DHT protocol (based on the Kademlia protocol) and a location based protocol. By utilizing these protocols, we provide several important features of a distributed intelligent system: scalability, flexibility, robustness, and location awareness.

2.5 Related Research

The following literature on distributed systems is relevant to our research and provides valuable information.

Ali [19] argued that “The cross-layer designs in sensor-nets lead to monolithic, vertically integrated solutions, which might work independently, but are not really useful for other research groups.” He proposed using P2P protocols over traditional sensor-nets to eliminate the need for proxy support and to enable flexible access to sensed data. A new protocol, Tiered Chord (TChord) based on Chord, was designed to fulfill the requirement of sensor-nets. Unfortunately, there was no practical implementation of this protocol in sensor-nets.

Barolli [20] used JXTA (an open source P2P protocol specification developed by Sun Microsystems) to realize a P2P platform. Song [21] used the Pastry protocol to realize a sensor data sharing system that was implemented on a desktop platform.
3: SYSTEMS ARCHITECTURE AND DESIGN

3.1 Introduction

We address here the following three research tasks to achieve the research objectives outlined in Chapter 1:

1. Set up a distributed intelligent system based on a P2P protocol in an embedded environment.

2. Realize the decentralization of sensing data publishing and querying.

3. Ensure sufficient data redundancy in the system so that even if some nodes of this system fail, the system's main functions and sensing data could be recovered from the remaining nodes.

According to Gruver et al., [22] the core requirements for a distributed intelligent system are a peer-to-peer communication infrastructure, a multi-agent architecture, and a means to provide distributed logistical support for the management and self-organization of system resources. Andrew [23] pointed out that coordination-based distributed systems played an important role in building distributed applications and inter-process communication was at the heart of all distributed systems. Thus, in our prototype system, two important issues should be resolved to accomplish these research tasks:

1. Communication and message transfer among different nodes.

2. Coordination of actions among all nodes.
An ad-hoc network was established and a series of communication packets were designed in order to deal with the first task. These message packets are representations of events in the event-driven architecture (EDA) autonomous model. They are responsible for finding node neighbors, publishing data, and querying data from remote nodes. The second task was accomplished by designing the working sequences of several critical procedures to ensure that this system could realize coordination among nodes.

3.2 Kademlia Protocol Modification

In contrast to traditional P2P file sharing applications, our system focuses on sensing information sharing. Thus, several modifications were made to the Kademlia protocol to ensure this protocol was suitable for sensing information publishing and querying.

1. All system node IDs were generated randomly as a 160-bit binary string. However, instead of using SHA-1 (a cryptographic hash function designed by the National Security Agency, to generate each key word's hash value), the value of sensing data key was directly converted from the original RFID tag ID value.

2. In contrast to traditional file sharing, published information (values) in this system is a complete detection event, which is defined as a quadruplet of the detected tag ID, sensor node ID, detection start time, and detection end time. Furthermore, any event whose duration is shorter than 10 seconds is considered to be noise. When a sensor node detects an event, it publishes it immediately.
3. When a node starts a query, the query arguments include the target’s tag ID, start time, and end time. The query result is a list of nodes that describes movement of the sequence of the target during the query time interval (tagID movement sequence: node A -> node C -> node E -> nodeA, from the start time to the end time).

### 3.3 Geographic Binary Tree

#### 3.3.1 Basic Design

To introduce location awareness into this system, we designed a protocol called the geographic binary tree. In this protocol, each node’s location (latitude and longitude) was used as an important property. In order to be consistent with the Kademlia protocol, each node is assigned a 160-bit node ID. Instead of generating the node ID randomly, it was generate as:

\[
\text{Hash key} = \frac{((\text{latitude} + \text{longitude}) \times (\text{latitude} + \text{longitude}) + \text{latitude} - \text{longitude})}{2}.
\]

The ultimate goal of this protocol is to generate a binary tree similar to the tree in Kademlia protocol. Instead of using the XOR metric, the Haversine formula is used to calculate the geographic distance between two nodes:

**Haversine formula [33]:**

\[
R = 6371; \quad // \text{earth’s radius, km}
\]

\[
\Delta\text{lat} = \text{lat}_2 - \text{lat}_1;
\]

\[
\Delta\text{long} = \text{long}_2 - \text{long}_1;
\]
\[ a = \sin^2(\Delta \text{lat}/2) + \cos(\text{lat}_1) \cos(\text{lat}_2) \sin^2(\Delta \text{long}/2); \]
\[ c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1 - a}); \]
\[ \text{Distance} = R \cdot c; \]

Each node in Kademlia stores contact information of its neighbors to realize query message routing. For each \( 0 \leq i < 160 \), every node maintains a list of \(<\text{IP address}, \text{UDP port}, \text{node ID}>\) triplets for nodes with distance between \( 2^i \) and \( 2^{i+1} \) from itself. In the geographic binary tree, nodes also hold records of their neighbors, except that the geographic distance is utilized to define the real-world neighbors for each node.

The main data structure is similar to Kademlia where \( k \)-buckets were maintained for different distance range. The result is shown in Figure 3.1, where \( k \) is set to five. Each layer represents a collection of nodes whose distances to the original node are between \( 2^i \) and \( 2^{i+1} \) meters.
The following table is an example of the geographic data where several nodes that were geographically close to the original node were selected:

<table>
<thead>
<tr>
<th>Node</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Distance to the original node (km)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original node</td>
<td>49.28347</td>
<td>-122.95034</td>
<td>0.0</td>
</tr>
<tr>
<td>N1 (min)</td>
<td>49.28347</td>
<td>-122.95035</td>
<td>0.0007253</td>
</tr>
<tr>
<td>N2</td>
<td>49.28241</td>
<td>-122.95329</td>
<td>0.2443</td>
</tr>
<tr>
<td>N3</td>
<td>49.28050</td>
<td>-122.95329</td>
<td>0.3935</td>
</tr>
<tr>
<td>N4</td>
<td>49.28047</td>
<td>-122.95815</td>
<td>0.6574</td>
</tr>
<tr>
<td>N5</td>
<td>49.497</td>
<td>-123.185</td>
<td>29.19</td>
</tr>
<tr>
<td>N6 (max)</td>
<td>-49.28347</td>
<td>122.95034</td>
<td>15390</td>
</tr>
</tbody>
</table>

Since the new hash formula was based on each node’s latitude and longitude, all node IDs were closely related to their location after the hash computation. In the following phase, the geographic distances were converted into a binary string format. By placing these nodes into the initial node’s
geographic tree, we obtained the resulting geographic binary tree shown in Figure 3.2. Each leaf node in this tree contains a $k$-bucket. The bucket capacity of each leaf is set to two.

![Figure 3.2: Binary tree based on geographic distance.](image)

As illustrated in the above example, all nodes were sorted automatically in the binary tree. The distance calculation and sorting may be time consuming, especially when dealing with huge amount of peers. A more flexible and dynamic sorting procedure is:

1. Calculate each node's distance to the original node.
2. Convert all distances into a binary string format called BinaryDist by using Java Type conversion (conversion between various Java variable types).
3. With BinaryDist that was generated in the previous step, all peers are put into the binary tree's leaf node in the following manner: if the leaf
node holds more peers than its bucket's maximal capacity, this leaf node will be split and two new nodes with their own buckets will be created. The peers contact information from the original bucket will be divided into new nodes. If the new buckets are still overloaded, we continue to split the leaf nodes until the entire tree’s leaves are not overloaded. Since BinaryDist already holds distance information, the leaves of the entire tree are automatically sorted. Furthermore, the left leaves are always closer to the original peer compared to their right siblings. In each leaf’s bucket, all peers are also sorted so that the closer peers are near to the top of the bucket and the further peers are at the bottom of the bucket.

4. This protocol will return a list of peers that are geographically near to the original node. Thus, nodes in the left sub-trees of the entire geographic binary tree should be the candidates.

3.3.2 Publishing and Querying

In contrast to DHT protocol design, initial nodes in a geographic binary tree will publish information to their geographic neighbors regardless of the content and type of the published content. Consequently, the method to effectively publish and query information could be an issue. In our design, a category tree will provide the solution to location target information. When a user wants to publish online resources to his neighborhood, assigning these resources with proper category information will speed up the users’ search process. When a person starts a query about a specific geographic
neighborhood, a list of available service categories will be obtained as shown in the Figure 3.3:

![Category tree diagram]

**Figure 3.3:** Category tree.

By selecting subcategories, users could locate their target in several steps. Furthermore, since each node in this P2P network will only be aware of a small subset of its neighborhood, information size of each node will not be excessive, which should help accelerate the search procedure.

### 3.3.3 Routing

Message routing in the geographic binary tree can be accomplished by traversing the entire routing table of one node and determining the geographically nearest $k$ nodes. The initial node could then send query information to these nodes and wait for a response from its neighbors. As the neighbors receive the query information, they will decide whether to respond directly or route these messages to their neighbors that are closer to the target
location. In contrast to the Kademlia protocol, this routing procedure is based on message flooding since it does not have strict control of network topology. Thus, the search result of this protocol is not guaranteed.

Figure 3.4: Routing in a geographic binary tree.

3.4 System Organization

To ensure that this prototype system achieves its research goals, an appropriate combination of hardware platform, embedded operating system, and network connection is critical to the system implementation. In our research, “Distributed Intelligent Node (DIN)” represents a hardware platform equipped with an embedded operating system, supporting software, wireless connection, and P2P protocol implementation.
3.4.1 Hardware Platform

Two hardware platforms were considered for the proposed system: (1) BeagleBoard-xM, a low-power, low-cost single-board computer produced by Texas Instruments and (2) TS-7300, a single board computer produced by Technologic Systems. A comparison of hardware configurations of these products [34], [35] is shown in Table 3.2.

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>RAM</th>
<th>On-board FPGA</th>
<th>Network</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS7300</td>
<td>ARM920T</td>
<td>32 MB</td>
<td>Cyclone FPGA</td>
<td>10/100 Ethernet, USB WiFi</td>
<td>PC/104</td>
</tr>
<tr>
<td></td>
<td>200MHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beagle Board</td>
<td>DM3730</td>
<td>512 MB</td>
<td>n/a</td>
<td>10/100 Ethernet, USB WiFi</td>
<td>GPIO</td>
</tr>
<tr>
<td></td>
<td>1GHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>USB</th>
<th>SD Card</th>
<th>Size</th>
<th>Serial Port</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS7300</td>
<td>2 USB 2.0</td>
<td>2 SD</td>
<td>152.4 x 121.92mm</td>
<td>DB9</td>
<td>$219</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(USD)</td>
</tr>
<tr>
<td>Beagle Board</td>
<td>4 USB 2.0</td>
<td>MicroSD</td>
<td>78.74 x 76.2mm</td>
<td>DB9</td>
<td>$149</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(USD)</td>
</tr>
</tbody>
</table>

The TS-7300 has several drawbacks, as described by Thomas [36]. It was designed and fabricated with hardware technology available in 2006. Hence, its specifications did not accommodate the rapid development and sophistication of embedded Linux systems. The Beagle Board-xM provides faster CPU processing speed and larger RAM. It is equipped with USB 2.0 and General Purpose
Input/Output (GPIO) interface, a generic pin on a chip whose input and output behavior can be controlled through software [37]. Because of these rich interface options, a range of peripherals can be accommodated. Furthermore, the Beagle Board-xM has better support for embedded operating systems. For these reasons, the BeagleBoard-xM was chosen as the DIN hardware platform.

3.4.2 Operating System

We considered three different embedded operating systems. The Android operating system is based on the Linux kernel. Google and other members of the Open Handset Alliance collaborated on Android’s development and release [38]. Angstrom OS is a Linux distribution of a variety of embedded devices [39]. ARM EABI Ubuntu is a specific version of Ubuntu that targets the ARMv7 and the Application Processor family (Processor Cortex A8, A9, and above) [40]. A detailed comparison among these operating systems is given Table 3.3:

<table>
<thead>
<tr>
<th></th>
<th>Android Froyo</th>
<th>Angstrom</th>
<th>ARM EABI Ubuntu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Support</td>
<td>Dalvik JVM</td>
<td>Cacao VM</td>
<td>openJDK-1.6.0</td>
</tr>
<tr>
<td>Serial Port Support</td>
<td>Android-serialport-api from Google Code</td>
<td>Librxtx-java, serial driver is unclear</td>
<td>Librxtx-java</td>
</tr>
<tr>
<td>GUI support</td>
<td>Android GUI</td>
<td>Console or X11</td>
<td>Console or Xfce</td>
</tr>
<tr>
<td>Database</td>
<td>SQLite database</td>
<td>SQLite database</td>
<td>SQLite database</td>
</tr>
<tr>
<td>WiFi adapter</td>
<td>Not available</td>
<td>Unclear</td>
<td>Working</td>
</tr>
</tbody>
</table>

Table 3.3: Comparison of operating systems.
In reference [36], the author states that an optimal distributed unit combination should be commercial Off-The-Shelf hardware, fully featured, optimized GNU/Linux with a better Java support. Due to ARM EABI Ubuntu’s strong support for these requirements, it was chosen for the DIN operating system.

### 3.4.3 Network Connection

The main features of an ad-hoc network is its self-organizing and adaptive nature, which means that network connections can be changed without the need for system administration. The term “ad-hoc” implies that the network may take different forms and it may be mobile, standalone, or networked. The ad-hoc network is has no need for fixed radio base stations, wires, or fixed routers [41].

There are several common features between P2P networks and ad-hoc networks [42], [43]:

1. Decentralized architecture: Both networks follow a decentralized architecture, whereby no centralized servers exist.

2. System autonomy: Decentralized architectures raise the requirement for system autonomy. An ad hoc network has to determine its own configurations such as addressing, routing, and position identification. In a P2P network, autonomy is required to realize proper operation and coordination of peers.

3. Transient connectivity: Both networks are dynamic, whereby nodes/peers join or leave the network unpredictably. In P2P networks, peers may connect and disconnect with the network at different times.
In ad-hoc networks, especially networks with mobile nodes, nodes could move in or out of the communication range.

4. Unequal individual resources: In both networks, nodes/peers do not always own equal resources. For example, a powerful desktop computer and several embedded devices could set up an ad-hoc network. Computer hardware configurations are different for peers in P2P networks.

Ad-hoc networks focus more on the network layer of the Open Systems Interconnection (OSI) model. Because P2P networks deal with application network issues, they focus on the application layer of the OSI model. Due to the latter features and concerns about network issues, the ad-hoc network was chosen for the physical implementation of the DIN network. The DHT was utilized to define a logical topology on application layer.

3.5 System Architecture Design

In our system, hierarchical system component architecture was utilized to help the developer focus on a specific function’s realization instead of being concerned about the upper and lower layers. Four layers were designed to achieve a system modularization as shown in Figure 3.5:
1. System-wide layer: Contains the functions of system logging, timing, and other general system tools. All other layers utilize functions in this layer.

2. Foundational layer: Network communication, database persistence, and peripherals control are implemented in this layer.

3. Core function layer: The algorithms and functions of Kademlia protocol and other functions based on P2P protocols are implemented in this layer.

4. User interface layer: Three user interfaces are provided in this layer to accommodate running environments.

Detailed descriptions of the Java classes in these layers and subsequent system domain diagrams will be provided in Chapter 4.
3.6 Message Packet Design

Many distributed systems and applications are built directly on top of the simple message-oriented model offered by the transport layer [23]. As a prototype, our system also selected message transfer as the principal communication model. To transfer information between nodes, information packets were introduced to encapsulate all the messages. In this system, these packets were transferred through an ad-hoc User Datagram Protocol (UDP) connection.

![Diagram of message transfer](image)

**Figure 3.6:** Message transfer.

Based on our research tasks and the Kademlia protocol, four types of packets were designed.

3.6.1 HelloRequest Packet and HelloResponse Packet

Hello packets were used to check the online status of other nodes. The initial node sends the HelloRequest packet to its neighbors in the routing table and waits for the responses from these peers. If it receives a HelloResponse packet from another node, it assumes that this node is still alive. If some nodes’
responses are overtime, those nodes are considered to be offline. The node then updates its own routing table accordingly in the next step. The design of the Hello Packets is shown in Figure 3.7:

<table>
<thead>
<tr>
<th>DDB Code (1 Byte)</th>
<th>HelloReq/Hello Res Code (1 Byte)</th>
<th>Sender NodeID (20 Bytes)</th>
<th>Sender IP (4 Bytes)</th>
<th>UDP Port (2 Bytes)</th>
<th>TCP Port (2 Bytes)</th>
<th>HELLO</th>
</tr>
</thead>
</table>

Figure 3.7: Hello packets.

For simplicity, ‘Sender Info.’ block is used to represent the information block of ‘Sender Node ID, Sender IP, UDP Port, and TCP Port’ as shown in Figure 3.8:

Sender Info.

Figure 3.8: Sender information.

3.6.2 FindNodeRequest Packet and FindNodeResponse Packet

The FindNode packets are used in the lookup process. Its main function is to obtain a list of nodes that are close to the target ID.

The target ID for lookup will be included in the FindNodeRequest packet. Nodes that receive this FindNodeRequest packet begin to find $k$ nodes that are
closer to the target ID and return these nodes to the initial node by sending back the FindNodeResponse packets.

![FindNode Packet Design](image)

Figure 3.9: FindNode packets.

### 3.6.3 PublishRequest Packet and PublishResponse Packet

Publish packets are used to publish detection events to other neighboring nodes from the initial node.

In the PublishRequest packet shown in the Figure 3.10, tag ID, start time, and end time are included and are sent to candidates that are generated by the lookup process. Receivers of this packet analyze it and store the detection event in their own databases. After recording the sensing information, receivers generate the PublishResponse packet that includes the successful save time and the target ID and send it back to the initial node as a confirmation.
3.6.4 QueryRequest Packet and QueryResponse Packet

The Query packets are used to get the movement sequence of a specific tag ID based on other nodes’ data.

Similar to the PublishRequest packet, querying target ID, start time, and end time are included in the QueryRequest Packet as shown in Figure 3.11. Receivers analyze these QueryRequest packets and find this tag ID’s movement sequence during the time gap in their own databases. They then generate QueryResponse packets, which include the tag ID, the event’s original node ID, start time, and send time and send it back to the initial query node.
3.7 Design of Main Functions Sequence

This system has three main functions. They work on the message packets that were designed in Section 3.6. Lookup process finds a list of nodes that are XOR near to the target. Publish process is used for publishing the detection event while Query process is used to query the movement sequence for a specific tag ID. In this Section, several Unified Modelling Language (UML) sequence diagrams are used to show the detailed working sequence for each function [44].

3.7.1 Lookup Process

In Lookup process, the main class obtains a list of neighbors based on the target ID from the local routing table. The main class then sends Hello packets to these neighbors to verify if they are still online. All neighbors that receive Hello packets generate response packets and send them back to the initial node. If some of the neighbors do not send the response packets, the main class of initial node could consider them as offline. It selects new candidates from the local routing table and sends out Hello packets again until the number of candidates reaches a system-wide parameter.

The main class then sends FindNodeRequest packets to these candidates and sets up a task listener to process all responses from these candidates. Candidates that receive the FindNodeResquest packet process this packet, find $k$ nodes that are close to the target ID from their local routing table, and generate the FindNodeResponse packets. After the task listener receives responses from its neighbors, it processes these packets and returns a list of
targets that are close to the target ID. This procedure continues until no closer candidates can be found from nodes that are discovered in the previous Lookup process. A complete lookup process sequence diagram is shown in Figure A.1 in Appendix A.

3.7.2 Data Publishing Process

When the main class of the proposed system detects the presence of an object, it generates an event slice that contains this sensor node ID, object tag ID, and the object presence start and end time. The main class starts a lookup procedure based on the tag ID and requires the Packet Factory to generate the PublishRequest packets at the same time. After the main class receives a list of destinations from the lookup procedure, it utilizes the UDPConnection Java class to send all PublishRequest packets to these destinations. At the same time, a task listener is set up to monitor all response packets for this publish task.

When other nodes receive the PublishRequest packet, they process this packet, extract published information, and check if they are already in their local databases and save the information if they are new. These nodes then generate PublishResponse packets that include a timestamp of the successful save and send it back to the initial publishing node.

When the publishing node’s task listener receives a response from target nodes, it processes the packet to confirm a successful save. After all responses are received, the task listener ends the publishing procedure. The complete working sequence is shown in Figure A.2 in Appendix A.
3.7.3 Data Querying Process

In order to query one object’s movement sequence, the main class will execute a query on the local database, create the initial movement sequence, and determine the missing time gap of this object. The main class then uses lookup process to get a list of candidate destinations. PacketFactory is used to get a list of QueryRequest packets. With the help of UDPConection, the main class sends the packets to these targets. Meanwhile, a task listener is set up to process the response packets.

When other nodes receive the QueryRequest packets, they process them to get the query arguments and start querying their local databases. If useful information exists, they generate QueryResponse packets accordingly and send them back to the initial query node. After the task listener obtains a response from other nodes, the initial node process these packets and keep updating the object’s movement sequence. The goal of this procedure is to generate a complete movement sequence of this object within the fixed period. The sequence diagram of the data query function is shown in Figure A.3 in Appendix A.
4: SYSTEM IMPLEMENTATION AND ANALYSIS

4.1 Introduction

In this Chapter, we describe the implementation and test of the system functions. This system was implemented in Java language. During the initial development phase, code was tested in an environment with three desktop computers that were connected through a router. The topology of this system during development phase is shown in Figure 4.1:

![Figure 4.1: Development phase system topology.](image)

After the main functions became available and stable, all systems were transferred to the test phase that had an ad-hoc environment with six nodes. Five nodes were running on Distributed Intelligent Node (DIN) as defined in Section 3.4 and one node was running on a laptop computer as a system-probing node. The test system topology is shown in Figure 4.2:
The laptop computer was used as a console to monitor the status of other nodes. In a practical deployment, this computer would be replaced by a DIN. All nodes work autonomously and the entire system is entirely distributed. In the development phase, mock commands were used at the laptop node to simulate RFID detections. After the development phase, RFID readers were attached to the DINs to validate system functions. In the final test phase, a motion detector and an IP camera were implemented on several nodes to verify the system’s flexibility.

4.2 Hardware Selection

As described in Chapter 3, the Beagle Board-xM was chosen as the hardware platform for all system nodes. A photo of the experimental node is shown in Figure 4.3:
Figure 4.3: Experimental node.

The experimental node is composed of two main components. The upper half is the Single Board Computer (SBC) BeagleBoard – xM, equipped with USB WIFI adapters to provide communication ability between nodes. The lower half is a RFID reader/writer connected through a USB to RS232 converter. Two different USB WIFI adapters were utilized. The comparison data are given in Table 4.1 [45], [46]:

<table>
<thead>
<tr>
<th>Name</th>
<th>Detection Range</th>
<th>Practical Issues</th>
<th>Transfer Speed</th>
</tr>
</thead>
</table>
| ASUS 802.11g 54Mbps WL-167g USB WLAN Adapter | Indoors: 330 ft, 100m
Outdoors: 1,150 ft, 350 m | stable             | 710 KB/s        |
| IOGEAR WIFI USB2.0 Adapter, 802.11g | 100-400 m, depending on surrounding environment | unstable         | 29 KB/s        |
The RFID device shown in Figure 4.3 is the APSX RW-210 RFID reader/writer. We also implemented the ThingMagic M5e RFID reader. The specification of these two RFID readers/writers is shown in Table 4.2 [47], [48]:

<table>
<thead>
<tr>
<th>Name</th>
<th>Operating Frequency</th>
<th>Data Rate</th>
<th>Read Frequency</th>
<th>Read/Write Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSX RW-210</td>
<td>13.56 MHz (HF)</td>
<td>19,200 bps</td>
<td>20/s</td>
<td>17.78 cm (10.16 cm radius cone shape)</td>
</tr>
<tr>
<td>ThingMagic Mercury5e</td>
<td>860-960 MHz (UHF)</td>
<td>9,600 bps</td>
<td>160/s</td>
<td>6.1 meters with 12 dBi antenna</td>
</tr>
</tbody>
</table>

### 4.3 Software Selection

Although the operating system provides the foundation for the system operation, supporting software was also introduced to facilitate the system development and debugging, as shown in Table 4.3:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>JVM</th>
<th>Database</th>
<th>Serial Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu 10.04 for ARM</td>
<td>OpenJDK 6.0</td>
<td>SQLite3, sqlite4java works as API libsqlite4java-linux-arm.so works as system driver.</td>
<td>Librxtx-java library</td>
</tr>
<tr>
<td>Remote Monitor</td>
<td>Remote Monitor Augment</td>
<td>Time Synchronization</td>
<td>Log File Server</td>
</tr>
<tr>
<td>Open-SSH</td>
<td>GNU Screen</td>
<td>NTP server</td>
<td>HTTP Server</td>
</tr>
</tbody>
</table>
Each DIN was loaded with Ubuntu 10.04 for the ARM platform with a command line user interface. The kernel and file system image of this Ubuntu operating system was provided by Embedded Linux Wiki (http://elinux.org/BeagleBoardUbuntu). Besides the operating system, additional software systems were utilized for the following requirements:

1. Java application running:

Because the application is realized in Java, a Java Virtual Machine (JVM) is required for the ARM platform. We have chosen OpenJDK 6 because it is a free and open source implementation of the Java SE 6 specification. Furthermore, Ubuntu has a very good support and its various binaries have passed the Sun Java SE 6 Test Compatibility Kit (TCK) suite [49].

2. Data persistence:

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine [50]. The SQLite was chosen because it supports Unix, OS/2, and Windows platforms, its database is open source, and it has been widely used in different computing environments, especially embedded devices. Its small code size and low resource requirement make it a perfect choice for our system development.

Sqlite4java is a wrapper to realize an easy utilization of SQLite database for Java applications [51]. System library libsqlite4java-linux-arm.so is also provided with this wrapper for running on the ARM platform in the Linux environment.
Several SQL scripts were executed in the SQLite console to create a database and related data table for our application as shown in Script 4.1.

**Script 4.1: Database SQL scripts.**

```sql
USE `tinydb`;

CREATE TABLE `event` (  
  `Id` int(11) NOT NULL AUTO_INCREMENT,  
  `tagID` varchar(192) NOT NULL DEFAULT '',  
  `originalNode` varchar(192) NOT NULL DEFAULT '',  
  `startTime` bigint(20) NOT NULL DEFAULT '0',  
  `endTime` bigint(20) NOT NULL DEFAULT '0',  
  PRIMARY KEY (`Id`)  
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8;
```

3. Device control:

Librxtx-java is a full Java CommAPI implementation. It is a library that provides serial and parallel communication for the JDK [52]. With the help of this library, communication with serial devices can be set up directly through Java API without considering the hardware details.

4. Remote monitoring:
Secure Shell (SSH) is a network protocol for secure data communication, remote command execution, and other secure network services between two computers that are connected through the network [53]. OpenSSH is a free version of the SSH connectivity tools. One drawback of OpenSSH is that when users log out from OpenSSH, all processes that originated remotely from OpenSSH will also be terminated. In order to ensure all the remote nodes could work properly even after the OpenSSH is disconnected, GNU Screen is introduced. It is a full-screen window manager that could control several processes through one physical terminal [54]. When Screen is called, it creates a new window in the original shell. Thus, the screen’s window and its related processes remain running when users logout from OpenSSH.

5. Time Synchronization:

The BeagleBoard-xM is not equipped with a real-time clock (RTC) that keeps track of the current system time [55]. Hence, this board will reset its system time to the date of its manufacture each time that it is switched on. In order to maintain a consistent system time throughout the entire network, a Network Time Protocol (NTP) server was set up in the laptop node. NTP is a protocol for synchronizing the clocks of computer systems over packet-switched, variable-latency data networks [56]. After switching on each node, the nodes will automatically connect to the NTP server to update their system times. The NTP server is set up with Scripts 4.2:
setup NTP time server

1. sudo apt-get install ntp
2. sudo cp /etc/ntp.conf /etc/ntp.conf.backup
3. sudo vi /etc/ntp.conf
4. add restrict 192.255.0.0 mask 255.255.255.0 nomodify notrap
5. sudo /etc/init.d/ntp restart
6. watch ntpq -p
7. sudo iptables -t filter -A INPUT -p udp --destination-port 123 -j ACCEPT
8. vi /etc/rc.local add /etc/init.d/ntp start
9. client: sudo ntpdate 192.255.0.1 to synchronize date information from NTP server
10. client: sudo dpkg-reconfigure tzdata to change the default system Time Zone setting.

6. RFID device setting:

A different hardware setting should be used for the RFID devices. The system settings for the APSX RW-210 and ThingMagic M5e RFID readers are shown in Table 4.4:
Table 4.4: RFID device system setting.

<table>
<thead>
<tr>
<th></th>
<th>Serial Port ID</th>
<th>Baud Rate</th>
<th>Data Bits</th>
<th>Stop Bits</th>
<th>Parity Check</th>
<th>Read Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSX RW-210</td>
<td>/dev/ttyUSB0</td>
<td>19,200</td>
<td>8</td>
<td>1</td>
<td>None</td>
<td>0xFA</td>
</tr>
<tr>
<td>ThingMagic M5e</td>
<td>/dev/ttyUSB0</td>
<td>9,600</td>
<td>8</td>
<td>1</td>
<td>None</td>
<td>{0x02, 0x22, 0x03, 0xE8}</td>
</tr>
</tbody>
</table>

7. GPIO setting:

A GPIO interface is used to read signals from external devices. Two tasks need to be accomplished to read external signals through the Beagle Board:

i. Enable BeagleBoard GPIO pin:

Das U-Boot (Universal Bootloader) is an open source, primary boot loader used in embedded devices [57]. U-Boot is used to copy the Linux kernel and root file system into the BeagleBoard memory and then boot the operating system. In order to enable the BeagleBoard GPIO pin, a U-boot source code named “u-boot-2010.12-rc3.tar.bz2” was downloaded and its “beagle.h” file was modified to enable GPIO pin 139 as shown in Script 4.3:
Script 4.3: U-boot code modification.

Original code: \texttt{MUX\_VAL(CP(MMC2\_DAT7), (IEN | PTU \setminus EN \setminus M4))}

Edited Code: \texttt{MUX\_VAL(CP(MMC2\_DAT7), (IEN | PTD \setminus DIS \setminus M4))}

In order to compile U-Boot, commands were executed as shown in

Script 4.4:

\begin{verbatim}
> cd u-boot-2010.12-rc3
> export CROSS_COMPILE=arm-unknown-linux-gnueabi-
> export PATH="/files/beagle/x-tools/bin:$PATH"

Now, configure and make U-Boot

> make distclean
> make omap3_beagle_config
> make
\end{verbatim}

After recompiling this U-Boot file and placing it back to the BeagleBoard, GPIO pin 139 is enabled to receive external signals.

ii. Read signal from GPIO:

To read signal from GPIO pin 139, Script 4.5 needs to be executed.
8. Routing Table:

Each node in this system owns a static file called initial.dat that stores initial contact information of its routing table. This file’s content was organized in the following format: <NodeID, IP Address, UDP port, TCP port>. Each node could create its initial routing tables by parsing this file.

4.4 System Development

4.4.1 Network Setup

As a wireless P2P system, network connection is very important to maintain the operation of the proposed system. We have chosen an ad-hoc mode to set up the system’s network environment and, hence, the Script 4.6 has to be written into each DIN’s /etc/rc.local file as an automatic running script to setup node’s ad-hoc connection.
Script 4.6: Ad-hoc network setting scripts.

```bash
sudo ifconfig wlanx down
sudo iwconfig wlanx mode ad-hoc
sudo iwconfig wlanx freq 2.412G
sudo iwconfig wlanx essid 'ideaAdhoc'
sudo ifconfig wlanx up
sudo ifconfig wlanx 192.255.0.y
// where wlanx is closely related with each node’s USB WIFI adapter, y is
the IP address we assigned to them
```

The network configured by Script 4.6 is an ad-hoc network running at a
frequency at 2.412 GHz. All nodes in this network share the same Extended
Service Set ID (ESSID) as ideaAdhoc.

4.4.2 User Interfaces

In order to provide a better user experience, three user interfaces were
developed: The main and fully functional UI is the command line interface. The
second UI is based on The Standard Widget Toolkit (SWT) [58]. The third is an
Android version UI that is still under development.

- Command Line Interface (CLI):

```
Please input your command
help
  which command would you like to know? e.g. help start.

help start
  start - start the whole system
SYNOPSIS
  start [DEVICE_PORT]
DESCRIPTION
  start the system, and begin the RFID scanning on a special port
  port format is [usb6/usb1/s1]

help publish
  publish - publish an object’s detection event
SYNOPSIS
  publish -t [TAGID] -s [START_TIME] -e [END_TIME]
DESCRIPTION
  A simulation of object detection
  time format is [yyyy/MM/dd mm:ss]
```

Figure 4.4: Command line interface.
• SWT GUI: this GUI was initially developed for testing purposes in a desktop environment. It holds the main functions of this system.

![SWT GUI](image1.png)

Figure 4.5: SWT graphical user interface.

• Android Froyo GUI: This GUI is under development. Due to the lack of network and serial driver supports, several main functions could not be utilized.

![Android GUI](image2.png)

Figure 4.6: Android graphical user interface.
4.4.3 Class Explanation

In Section 3.5, four main layers of the proposed system were described. A detailed explanation of the main Java classes for each layer is given in following Sections.

4.4.3.1 Core Function Class

The core functions of this system are realized through Java classes. These classes provide the system with the ability of network communication, multi-threading, and the Kademlia P2P protocol implementation.

DDB: This is the main class and entrance of the entire system. It is in charge of all other components’ initialization, such as FakeData (for data simulation), UDPConnection, RoutingTable, LookUp, Publish, Query, and PacketProcessor. It analyzes the incoming packets and decides the execution of subsequent actions based on the type and information of incoming packets. It also maintains the data exchange with the system UI.

Bootstrap: This class is used to update a node’s initial routing table. By sending out Bootstrap request packet and analyzing the incoming response packets, one node could discover new nodes in this network. This function improves system redundancy, speeds up message delivery, and ensures system robustness.

PacketFactory: This factory class is in charge of generating all types of packets. Detailed explanations of packet types have been given in Section 3.6.

TaskListener: A list of listeners is utilized to monitor the progress of various tasks in the system. These listeners are in charge of the system status
update, process ending, resource recovery, routing table update, and other related tasks.

LookUp: This is the core function of the entire system. LookUp process returns a list of nodes that are XOR near to the querying ID. By running LookUp process, a node’s routingTable is updated in real time.

Publisher & Query: This is based on the basic system components, especially LookUp procedure, Publisher & Query components that are in charge of publishing sensing data to remote nodes, and querying data from remote nodes. In the beginning of a node’s querying or publishing activities, LookUp process is executed to obtain a list of destinations. The initial node can then publish its new sensing data to these destinations or send query to these nodes through UDPCConnection. TaskListener monitors the progress of these activities.

The listed Java classes define the basic data structure for the Kademlia protocol.

Int160: This class defines the unique ID for all nodes and objects in the system. It provides other components with methods including random ID generation and RFID tag ID conversion.

EventSlice: This is the definition of the detection event in this system (an object’s presence in one sensing node’s detection range in a time interval).

Peer, Bucket, Node, RoutingTable: These classes are components of the system’s routing table. Peer refers to each running machine in the network. Node refers to the node in the local routing table’s binary tree. Bucket refers to the $k$-bucket at each leaf in the binary tree. RoutingTable defines the core data
structure that is used to store a node’s neighbors. It is a binary tree and all leaf nodes record this node’s neighbors. The location of this leaf is determined by the XOR distance with this node.

Timer: In a self-organized system, Timer realizes system autonomy. It maintains a TaskList that contains all Tasks that implement the Java Runnable interface. A private class TaskExecutor that extends Thread class executes all these tasks at a pre-defined time interval.

4.4.3.2 System-wide Class Explanation

System wide classes provide functions that maintain system’s operation, such as system logging, HTTP server, and data structure converting.

FileLogger: This is a singleton class that maintains a record of system events. Other system classes may use this class directly to record important running information in the log file.

HTTPserver: This class provides a fully functional HTTP server running at port 20020 named NanoHTTPD [59]. It enables other users to access the log files and surveillance pictures taken by the IP camera.

Convert, Utils: Convert class provides all important data structure converting functions in this system. Utils class provides useful methods such as getLocalIP (return local system’s IP address) and routingTablePrint (print local routing table’s binary tree by level).
4.4.3.3 Foundational Class

The following classes maintain communication with various external resources:

Persistence: This class maintains communication with the SQLite database. It maintains database operations, such as connectDatabase, saveEvent, hasEvent, and queryEvent.

ApsxReader: This class maintains the communication with the APSX RW-210 RFID reader.

SimpleReader: This class maintains the communication with the ThingMagic m5 RFID reader.

EventFilter: Based on the RFID reader’s detection information, this class generates EventSlice, the detection event. The main function of this class is to continuously update each event’s ending time.

ArrayFilter: Based on the EventSlices generated by EventFilter, this class will filter out noises (events that last no longer than 10 seconds). It then saves these events in local database and triggers the Publish process to publish them to the neighbors.

UDPCConnection: This class helps maintaining network connection. It manages a listening thread along with the start of the entire system. It also is in charge of sending data packets through a specific communication port.
4.4.3.4 CLI/GUI Class

CLI/GUI classes enable users to monitor each node’s status. This interface may also be used to simulate various detection events during the system development phase.

Text2Command: This class translates the user’s inputs to system commands in CLI, including routing table’s loading, RFID reader selection, system bootstrap, mock detection publishing, and information querying.

NodesTab, PublishTab, QueryTab: GUI, based on the SWT and Android Froyo, provide users with the real-time information from routing table, functions of mock publishing, and information querying.

4.4.4 Diagrams of the System Domain

The UML domain diagrams show how classes work together in various system phases.

4.4.4.1 System Initialization Domain

In the system initialization phase, the main system class DDB turns on the main components of this system, loads the system’s initial routing table from initial.dat file, ensures that the system is ready to read/write sensing information, and sends/receives communication packets between nodes. The domain diagram for this phase is shown in Figure A.4 in Appendix A.

4.4.4.2 System Publishing Domain

In the publishing phase, class Publisher works as the entry point of the publishing function. In the newly created instance PublishDataTask, Kademlia’s
core function Lookup is executed and a list of candidates is made available for the next step. After sending the PublishDataRequest packets to all candidates, this publishing procedure is completed. The domain diagram of system publishing is shown in Figure A.5 in Appendix A.

### 4.4.4.3 System Communication Domain

![Diagram of the system communication domain.](image)

**Figure 4.7: Diagram of the system communication domain.**

Two classes are extended from Java Thread class as the main implementation of communication between different nodes. ListenThread keeps monitoring incoming packets, while SendThread is in charge of sending packets to their destinations. DatagramChannel is added for non-blocking UDP applications in Java 1.4. In UDP mode, a single datagram socket processes
requests from multiple clients for both input and output. DatagramChannel makes this non-blocking so that methods return quickly if the network is not immediately ready to receive or send data [60].

4.5 System Analysis

4.5.1 System Environment Preparation

The experimental system topology is shown in Figure 4.8. This topology is used to define the initial routing table for each DIN.

![Experimental system topology](image)

**Figure 4.8:** Experimental system topology.

In the test phase, six nodes were deployed as shown in Figure 4.9:
4.5.2 Experimental Results

The running status and output of the main system functions describe the experimental results. Due to the different performances of the IOGEAR and ASUS USB WIFI adapters, nodes N3 and N5 were chosen as the main sensing nodes. They are equipped with the ASUS WIFI adapter. Nodes N4 and N6 were used as internal nodes with the IOGEAR WIFI adapter. N1 was deployed on the laptop computer to work as the ad-hoc network establishment node, NTP server, system monitor, and debug center. From one stochastic node’s command line
user interface, after retrieving complete routing information from others nodes, a node’s routing table binary tree is shown in Figure 4.10:

![Binary tree based on XOR distance.](Image)

**Figure 4.10: Binary tree based on XOR distance.**

### 4.5.2.1 Neighbor Checking Function

An important system function is to periodically check each neighbor’s status and ensure that nodes in the local routing table are accessible. Thus, in a fixed period, each node selects several nodes whose last response time is older than a system-wide parameter. It will then send out HELLO Request packets to these nodes and update the local routing table based on the responses from these nodes.
N2 received HELLO Request packet from N4 and N5 is shown in Figure 4.11. Since N2 was still online, it created the HELLO Response packet and sent it back to their original request nodes.

Figure 4.11: N2 response with HelloResponse packet.

Figure 4.12: N5 update routing table.
After sending out HELLO Request packets, N5 waits for the reply and updated each node’s status accordingly. Since the N6’s response was overtime, this node was removed from the N5’s routing table.

4.5.2.2 Bootstrap Function

The Bootstrap function is utilized when a new node joins the existing network. In order to create a local routing table, this new node needs to know at least one node that already exists in the network. By sending BOOTSTRAP Request packet to this existing node, the new node gets to know a subset of this network based on the response from the existing node.

```
bootstrap
| received packet from other peer called: /192.255.0.2:4666
  Begin Listening----------------
  Begin analysing the packet----------
Hoho, a Hello Req Packet---------```

Sending Hello response packet  ________________
End Sending Hello Response to/192.255.0.2:4666------------------
I am bootstraping---------!!

End Sending Bootstrap Request to/192.255.0.3:4666----------------
End Sending Bootstrap Request to/192.255.0.2:4666------------------

received packet from other peer called: /192.255.0.3:4666
Begin Listening----------------
Begin analysing the packet----------

Haha, a bootstrap Res Packet---------```

received packet from other peer called: /192.255.0.2:4666
Haha, a bootstrap Res Packet---------```

Begin Listening----------------
Begin analysing the packet----------
BootStrap completed

Figure 4.13: N1 bootstrap procedure.

N1 had N2 and N3 in its routing table in the beginning, as shown in Figure 4.13. In order to bootstrap, N1 sends BOOTSTRAP Request packets to N2 and
N3. The bootstrap procedure terminates after responses from N2 and N3 are received and after new nodes are added to its routing table.

Figure 4.14: N2 and N3 responses to BOOTSTRAP Request packet.
After N2 and N3 receive the BOOTSTRAP Request packets, both nodes send back to N1 the BOOTSTRAP Response packets with the nodes contact information they selected from their own routing tables.

Figure 4.15: Routing table of N1 after bootstrapping.
After the completion of BOOTSTRAP, the overall structure of N1’s local routing table is shown as a binary tree in Figure 4.15. As shown, new nodes N4 and N5 are added into N1’s routing table.

In conclusion, by using BOOTSTRAP, each new node learns a subset of the entire network through one existing node. Due to the small number of system nodes in our experiment, we did not consider permanently recording information of these new nodes into each node’s initial.dat file. Otherwise, each node would know almost the entire experimental network from the initial phase.

4.5.2.3 Publishing Function

Sensing nodes N2, N3, N5, and N6 were deployed near four main entrances of the laboratory. Each node continuously scans their areas. When they detected the presence of an object, a detection event is created that contains this object’s tag ID, start time, and end time of presence. This node keeps updating this event’s ending time as long as this object is still in the detection range.
In this experiment, subjects are asked to carry a RFID tag and walk through the laboratory in the sequence N2, N3, and N5 as shown in Figure 4.16. Node N2 generates the first detection event that started at 2011-06-27 15:16:33 and ended at 2011-06-27 15:17:03. This node publishes the information to N1 and N4, as shown in Figure 4.17.
Figure 4.17: N2 publishes first detection to N1 and N4.

N3 generated the second detection event that starts at 2011-06-27 15:17:15 and ends at 2011-06-27 15:17:46. This node then publishes the information to N1 and N2, as shown in Figure 4.18.
Figure 4.18: N3 publishes second detection to N1 and N2.

N5 generates the final event that starts at 2011-06-27 15:21:41 and ends at 2011-06-27 15:22:01. This node then publishes the information to N1 and N2, as shown in Figure 4.19.
In conclusion, we found that during the publishing procedure all nodes successfully generated the detection events and sent the PUBLISH Request packets to the candidates that were selected by the Lookup procedure.

4.5.2.4 Querying Function

Several specific command formats are designed in command line interface to simplify the query test procedure.

Command such as “query -t 0000302fbb0200104e0ead7 -s 2011/06/27 15:15:00 -e 2011/06/27 15:25:00” are used to query an object’s movement sequence between 15:15:00 and 15:25:00 on June 27, 2011, where ‘-t’ indicates the following string is the querying tag ID, ‘-s’ represents the detection’s start.
time, and ‘e’ is the detection event’s end time. The query result from N1 is shown in Figure A.6 in Appendix A.

The query result from node N6 is shown in Figure 4.20:

![Figure 4.20: Query result from N6.](image)

Every node may get the right movement sequence of this object in the querying time range. During our experiment, N1 and N3 were shut down on purpose to check system robustness. Then N3 was back online and Table 4.5 shows the querying procedure of the remaining nodes.

<table>
<thead>
<tr>
<th>Query Node</th>
<th>Lookup Nodes</th>
<th>Final Query Target Nodes</th>
<th>Lookup Nodes</th>
<th>Final Query Target Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1 and N3 are dead</td>
<td>N3 is back online</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.5.2.5 Lookup Performance

Lookup is the core function of the Kademlia protocol and its performance is one of the performance indicators of this system. The main output of the function is a list of nearest neighbors of a target ID.

In this experiment, we designed a simulation to avoid massive application deployment. Instead of utilizing physical computers, a list of predefined routing tables with simulated information was created. If a node gets a list of candidates from its local routing table, the IP addresses of the corresponding candidates are converted into their table names (e.g., for candidate with IP address 192.255.0.4, system checks the file table4.yaml directly instead of sending messages to 192.255.0.4). Furthermore, instead of the 160 bit node ID, a 16 bit node ID was introduced to simplify the testing procedure.

We finally tested the worst scenario for lookup function where each neighbor only hold nodes that were one bit closer to the target ID. The output of this lookup procedure is shown in Figure 4.21.
Figure 4.21: Worst case lookup for 16 bit ID.

The lookup target ID shown in Figure 4.21 is 1011010111100100 and the original querying node only has contact information about node ID 111101011100100 whose IP address is 192.255.0.2. A 16 hops lookup is shown. One node was given additional information about other nodes. In this experiment, information about candidate 1011010110100100, whose IP address is 192.255.0.10, was added to the routing table of candidate 192.255.0.4. The lookup procedure changes as shown in Figure 4.22. As expected, the lookup procedure was faster due to the additional contact information.
In the next phase, a list of target IDs with different numbers was created randomly and their lookup message numbers were collected as shown in Figure 4.23.

![Figure 4.23: Lookup messages for different target ID.](image-url)
The x-axis shown in Figure 4.23 represents the number of target IDs and the y-axis represents the final lookup messages that were sent. As shown, the number of lookup messages for an increasing number of target IDs is nearly constant, which implies that the system has a good scalability.

**4.5.2.6 Message Transfer**

Since a majority of the actions are triggered by messages sent from other nodes, message transfer is another performance indicator of the system. In this test, a linear topology is created and all nodes in this topology were only aware of neighbors that were one hop away (e.g., N3’s routing table only contained node N2 and N4). By using Bootstrap function, we can find a rough time that a message transfers from N1 to N6. The result is approximately 8.6 seconds.

![Figure 4.24: Linear testing topology.](image)

In order to make a comparison, we tested the original testing topology and linear topology. Node N1 sends a message of 35 bytes. Based on the responses from neighboring nodes, this message should be sent to N6.
Due to the network redundancy included in the original topology, N1 can reach N6 much faster than with the linear topology as shown in Figure 4.25. During the experiments with the linear topology, it was difficult to successfully send messages more than 60 times to N6 without failure due to the unstable WIFI adapter and delay of message transfer among nodes. However, the success rate is much higher because the distributed topology provides better system robustness.

4.5.2.7 System Flexibility Test

One benefit of a distributed intelligent system is the functional flexibility that it supports. Without reprogramming the system, new functions may be introduced through updating and adding nodes. By attaching the IP camera and
motion detector to the DIN, the distributed surveillance ability is added to the system. When an RFID reader discovers a new detection event, it sends a TakePictureRequest packet to the node with the IP camera, as shown in Figure 4.26. This camera node takes the picture, saves it in a public folder, and sends its URL address back to the original request node, as shown in Figure 4.27.

![DIN with IP camera](image)

**Figure 4.26:** DIN with IP camera.

![Picture's URL address](image)

**Figure 4.27:** Picture's URL address.
The implementation of a real-time surveillance node with the AXIS 210 network camera is shown in Figure 4.26. By setting an IP address with a different netmask, one DIN was assigned with two IP addresses: 192.255.0.5 and 192.255.1.5. Meanwhile, the IP camera was assigned with a static IP address 192.255.1.7. In this configuration, this node may receive live video and images from the IP camera without interfering with its original functions.

Using BeagleBoard’s GPIO interface, we also added a motion detector was added to this system, as shown in Figure 4.28. When a subject appears in the detection area, the motion detector sends a signal to the BeagleBoard’s GPIO interface. The application in the BeagleBoard then sends a TakePictureRequest packet to the camera node and camera node takes a picture to record this subject.

Figure 4.28: DIN with motion sensor.

A surveillance system with different peripherals is shown in Figure 4.29.
The event sequence is:

1. A person enters the Lab.
2. The motion detection node discovers the person and sends a TakePictureRequest packet to the camera node.
3. The camera node takes the image, saves it in its public folder, and sends the URL address of the image back to the original node.
4. The person takes an object with an RFID tag on it.
5. The RFID reader node discovers the object and sends another TakePictureRequest packet to the camera node.
6. The camera node takes a second image and sends its URL back to the RFID node.
The distributed surveillance system has the following advantages compared with a Client/Server surveillance system:

1. Fully distributed: The system has no central server for intruders to attack and it can continue to operate even if some nodes fail.

2. Ad-hoc connection: Cables are not needed to set up the network. The small size of the DIN ensures this system may be deployed in almost any location.

3. Functional flexibility: Different types of peripherals could be connected to introduce new system functionalities through USB and GPIO interfaces.
5: OPEN ISSUES AND FUTURE DEVELOPMENT

5.1 Introduction

In this thesis, we have developed a distributed intelligent system that integrates a single board computer, an embedded operating system, an ad-hoc network connection, different peripheral techniques, and P2P protocol implementation. This prototype system enabled us to realize a DIS with the features of strong system robustness, good scalability, flexible functionalities, and convenient deployment to make full use of available computing resources and to deal with a dynamic system environment.

Our research encountered several issues:

- unstable network connections
- poor database performance
- complex settings of peripherals.

We discuss here these issues and present what we believe should be a better DIS system.

5.2 Open Issues

5.2.1 Ad-hoc Network

In order to allow every node freely join and leave an ad-hoc network, at least one node has to establish this network manually in the beginning. This is incompatible with the system’s autonomy requirement.
Another issue is the poor performance of WIFI adapters, which resulted in unstable connections between nodes. Frequent loss of the wireless connection occurred during the experiments.

In order to re-establish a connection, we employed the “ping” command to test the reachability of non-responsive nodes, as shown in Script 5.1. Sometimes there was no reply, which implied that these nodes were unusable. Furthermore, the system development and testing were accomplished using the same private network. To communicate with devices from other networks, techniques such as network address translation (NAT) need to be introduced.

**Script 5.1: Command Ping is used to reconnect.**

```
ssh: connect to host 192.255.0.4 port 22: No route to host
PING 192.255.0.4 (192.255.0.4) 56(84) bytes of data.
  From 192.255.0.1 icmp_seq=1 Destination Host Unreachable
  From 192.255.0.1 icmp_seq=2 Destination Host Unreachable
  From 192.255.0.1 icmp_seq=3 Destination Host Unreachable
  From 192.255.0.1 icmp_seq=4 Destination Host Unreachable
  From 192.255.0.1 icmp_seq=5 Destination Host Unreachable
  From 192.255.0.1 icmp_seq=6 Destination Host Unreachable
64 bytes from 192.255.0.4: icmp_req=8 ttl=64 time=1.59 ms
  64 bytes from 192.255.0.4: icmp_req=9 ttl=64 time=0.981 ms
  64 bytes from 192.255.0.4: icmp_req=10 ttl=64 time=1.13 ms
  64 bytes from 192.255.0.4: icmp_req=11 ttl=64 time=0.784 ms
```

5.2.2 SQLite Database Performance

The main problem with the SQLite database is the lack of appropriate database tools for the Java language. During the implementation and system test, only one tool, still under development, was available for the ARM platform [51]. Furthermore, each database connection could only be utilized by the thread that
created it. Thus, many database connecting and closing actions had to be executed, which greatly lowered the performance of database operation.

5.2.3 GPIO Utilization

In order to make effective use of GPIO interfaces on the BeagleBoard, the U-boot of the operating system had to be modified and recompiled to enable some specified pins. In the developed system, GPIO interfaces were used to connect the motion detector. Although the motion detector had a special signal output, it could not be used directly with the Beagleboard since the output signal did not meet GPIO interface requirements. To detect this signal, a specially designed circuit was utilized to work as a bridge between these two devices, which lowered the system’s flexibilities with peripherals.

5.3 Future Development

5.3.1 Network Improvement

In the developed prototype system, network communication was accomplished through ad-hoc connections. In future developments, a hybrid system with wireless and wireline connections should be designed, especially when dealing with legacy systems.

To resolve issues with ad-hoc connections, Super WIFI could be a promising solution. This is a wireless networking proposal that the FCC plans to use for long-distance wireless Internet connections based on lower frequency (white spaces). This technique is close to commercial deployment. The maximum distance that can be obtained between the transmitter system and the receiver
system is 360 m (1180 ft) [61]. With Super WIFI, long distance and stable wireless connections could become a reality and this stable connection could be the foundation for future distributed systems.

With the implementation of IPv6, a more flexible and scalable network will be available. As an enhancement and alternative to IPv4, IPv6 provides $2^{128}$ (approximately $3.4 \times 10^{38}$) number of IP addresses for users and devices. Each person could obtain $4.86 \times 10^{28}$ addresses if there were 7 billion people on earth [62]. In the future, a 128 bits IP address could provide the universal and unique ID for a person, a computer, or an intelligent sensor in the network [63]. In this environment, a unique ID in P2P network could be replaced directly by IPv6 address.

5.3.2 P2P Improvement

Kademlia, which was chosen as the P2P protocol realization in our system, also has certain stability and efficiency disadvantages [24]. The availability of P2P protocol could be improved with a middleware that contains all the main P2P protocols. This middleware could help to decouple P2P routing from other layers as shown in Figure 5.1. Given the protocol choice and querying targets, such middleware could generate a list of candidate peers. These candidates could then be used by higher-level applications of this system.
With network improvements, especially the utilization of IPv6, this P2P middleware could even be moved from the application layer to the network layer of the OSI model. In this case, the P2P middleware could receive data packets, analyze the IPv6 addresses, and route these packets to the destination network directly based on DHT protocols. In this scenario, the next generation network could be a fully distributed, flat network environment where message routing is accomplished through DHT protocols that are implemented in hardware analogous to the current Network Interface Controller (NIC) and decentralized router functions in each network device. Thus, each device in this distributed network could operate as a client, a server, and a router at the same time, as shown in Figure 5.2.
5.3.3 Hardware Improvements

During the system development and test, the BeagleBoard-xm provided computing performance and storage space. In future developments, the following improvements should be considered:

- **Onboard WIFI, sensors, and GPS:** They will be a great enhancement to the current DIN. Wireless connection is an essential part for future systems. It will enhance the platform with uninterrupted network connection. With general sensors and GPS module, this platform could be easily configured to adapt to task requirements.

- **Onboard Field-Programmable Gate Array (FPGA):** With FPGA platforms and future developments based on dynamic partial
reconfiguration [64], [65], DINs will be able to realize different functions at the hardware level, which will provide faster processing and improved flexibility.

- General peripheral adapter: This adapter will be in charge of communication with different devices, especially low-level peripherals that do not have USB or serial interfaces. As an extension of the DIN platform, it could provide more flexible peripheral options.
- Lower power consumption and price: Power consumption and price affect the commercialization of distributed intelligent systems.

5.3.4 Data Persistence and Security

Along with the rapid development of embedded devices, better database accessing tools should be provided. Migrating the system from Java to C or C++ speed up the system’s processing speed, especially with the C/C++ SQLite library.

Current P2P protocols, particularly DHT-based protocols, may suffer from security issues from malicious nodes, which send back erroneous data objects to the lookup queries [4]. To deal with this issue, cryptographic techniques should be considered to protect sensitive information. Furthermore, by adding up each node properties such as reputations based on their former behavior could eliminate the influence of malicious nodes.
5.3.5 Intelligent Autonomous Model

We employed an event driven architecture (EDA) to achieve system autonomy. Thus, the overall system was operating as a message driven model. To update or add new system functions, new system messages should be added. However, this architecture is not sufficiently intelligent or functionally flexible since new message packets need to be designed to realize new functions. In the future, a pre-configured node with inbuilt local goals to define system execution should be introduced. With such autonomous nodes, this system could become self-regulated without global control.

5.3.6 Potential Application

Centralized system architectures have significant disadvantages especially when dealing with dynamic environments. The developed distributed architecture is not intended to entirely replace the centralized system. However, we believe that DIS could adapt itself better in future network environments.

Potential applications of our system are dynamic networks for intelligent transportation systems, intelligent building/environment monitoring, decentralized online search, and communication and advertisement that require strong system flexibility, scalability, and robustness.

5.4 Final Evaluation and Summary

Since the birth of the computer, centralization has dominated every aspect of computing. However, centralized architectures suffer from massive duplication of information, lack of robustness, severe privacy issues, waste of client-side computing resources, and lack of flexibility and scalability. Nature demonstrates
an alternative way of thinking through social insects. Along with rapidly
developing hardware and software, intelligent devices of small size, low price,
low power consumption, and powerful computing resources are becoming
available. By empowering them with the ability of peer-to-peer communication,
autonomous decision-making, and coordination between nodes, a distributed
intelligent system will serve as a better alternative for system design,
development, and application.

In this thesis, we proposed a distributed intelligent prototype system based
on an off-the-shelf low priced, high power single board computer, Kademlia DHT
P2P protocols, and ad-hoc communication. We presented a detailed system
design and consideration of hardware, operating system, supporting software,
P2P protocol modification, and realization. A fully functional application
architecture design, function partition, working flow design, and detailed system
class design were given. The distributed system functions were tested in a
practical environment, system robustness was demonstrated in some severe
environments, and its flexibility was demonstrated by adding external devices.

Distributed intelligent systems based on P2P protocols provide a useful
paradigm for system design and implementation. In the developed system,
collaboration determines the success of the task, sharing among participants
provides unlimited computing resources, decentralized design endows the
system with scalability, and a certain degree of redundancy ensures the safety of
information. As Friedman states, “the dynamic force in Globalization 3.0 is the
newfound power for individuals to collaborate and compete globally” [66]. We
could say, the force in Globalization 4.0 could be P2P based intelligent devices,
where the world is shrinking into small devices, so that the Internet of Things and ubiquitous computing could become a reality.
APPENDICES

Appendix A

Figure A 1: Lookup process
Figure A.2: Data publishing process
Figure A.3: Data querying process
Figure A.4: Diagram of system initialization domain
Figure A.5: Diagram of system publishing domain
Figure A.6: Query result from N1
REFERENCE LIST


