A Coordination Model and Framework for Developing Distributed Mobile Applications

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By

Xiaodan.Li

Permission to Use

Abstract

How to coordinate multiple devices to work together as a single application is one of the most important challenges for building a distributed mobile application. Mobile devices play important roles in daily life and resolving this challenge is vital. Many coordination models have already been developed to support the implementation of parallel applications, and LIME (Linda In a Mobile Environment) is the most popular member. This thesis evaluates and analyzes the advantages and disadvantages of the LIME, and its predecessor Linda coordination model. This thesis proposes a new coordination model that focuses on overcoming the drawbacks of LIME and Linda. The new coordination model leverages the features of consistent hashing in order to obtain better coordination performance. Additionally, this new coordination model utilizes the idea of replica mechanism to guarantee data integrity. A cross-platform coordination framework, based on the new coordination model, is presented by this thesis in order to facilitate and simplify the development of distributed mobile applications. This framework aims to be robust and high-performance, supporting not only powerful devices such as smartphones but also constrained devices, which includes IoT sensors. The framework utilizes many advanced concepts and technologies such as CoAP protocol, P2P networking, Wi-Fi Direct, and Bluetooth Low Energy to achieve the goals of high-performance and fault-tolerance. Six experiments have been done to test the coordination model and framework from different aspects including bandwidth, throughput, packages per second, hit rate, and data distribution. Results of the experiments demonstrate that the proposed coordination model and framework meet the requirements of high-performance and fault-tolerance.

Contents

Permission to Use **i**

Abstract **ii**

Contents **iii**

List of Tables **v**

List of Figures **vi**

1 Introduction **1**

2 Problem Definition **6**

3 Literature Review **9**

3.1 Coordination Models and Languages . . . 9
3.1.1 Linda . . . 10
3.1.2 LIME - Linda In a Mobile Environment . . . 12
3.2 Peer-to-Peer Network . . . 16
3.2.1 P2P Brief Introduction . . . 16
3.2.2 History of P2P . . . 17
3.2.3 Taxonomies of P2P architecture . . . 18
3.3 Connection Technology . . . 19
3.3.1 Wi-Fi Network . . . 20
3.3.2 Wi-Fi Direct . . . 23
3.3.3 Bluetooth . . . 31
3.4 REST and CoAP . . . 37
3.4.1 REST . . . 37
3.4.2 CoAP . . . 38
3.5 Distributed Hash Table and Consistent Hashing . . . 40
3.5.1 Distributed Hash Table . . . 40
3.5.2 Consistent Hashing . . . 40
3.6 Summary . . . 43

4 Solution and Architecture **45**

4.1 Purpose and Motivation . . . 46
4.2 General Introduction . . . 47
4.3 Coordination Model . . . 49
4.3.1 Organization of Tuples and Tuple Spaces . . . 49
4.3.2 Operations . . . 50
4.3.3 Engagement and Disengagement . . . 51
4.3.4 Fault Tolerance . . . 52

5 Implementation **54**

5.1 Design . . . 54
5.2 Coordination Layer . . . 56
5.2.1 Consistent Hashing . . . 56
5.2.2 The SystemRuntime Object . . . 58
5.2.3 Replica Management . . . 60
5.2.4 Tuple Spaces Management . . . 63

5.3 Message and Service Layer . . . 63
5.3.1 Service/Resource Management . . . 64
5.3.2 CoAP Message Management . . . 64
5.4 Networking Layer . . . 65
5.4.1 Wi-Fi Direct . . . 66
5.4.2 Bluetooth Low Energy . . . 68
5.5 The Framework Core . . . 68

6 Experiments **70**
6.1 Experiments Environment and Bandwidth Test . . . 71
6.2 Basic Throughput . . . 72
6.3 Throughput of the Coordination Framework . . . 73
6.4 Engagement and Disengagement . . . 74
6.5 Data Distribution . . . 76

7 Summary and Future Work **77**
7.1 Summary . . . 77
7.2 Future Work . . . 79
7.2.1 Engagement and Disengagement . . . 79
7.2.2 Replica Mechanism . . . 80
7.2.3 SystemRuntime Synchronization . . . 81
7.2.4 Support More Language . . . 81

List of Tables

3.1 Development History of Wi-Fi Network 21
3.2 Detail Specification of Wi-Fi Direct 27
3.3 Bluetooth History Versions 31
3.4 Summary of the Literature Review 44

6.1 Devices Configuration 71

List of Figures

1.1 Multiple Devices Work Together for a Single Application 1
1.2 Search Interests Trends on Mobile Cloud Computing 2
1.3 KPCB Internet Trends 2016 2
1.4 Single Platform Users' Share 2

2.1 Coordinations of Distributed Mobile Applications 6

3.1 Fundamental Operations in Linda 11
3.2 Transiently Shared Tuple Space in LIME 13
3.3 Topology of the Tuple Space in LIME 13
3.4 Tuple Space in Linda vs. Transiently Shared Tuple Space in LIME 14
3.5 Examples of P2P network and classic Client-Server Network 17
3.6 An Example of An Unstructured P2P Network 19
3.7 An Example of a Structured P2P Network 20
3.8 Connections Comparison between Conventional Wi-Fi and Wi-Fi Direct 24
3.9 Global Wi-Fi Direct Device Shipment 25
3.10 An Example Wi-Fi Direct Network 26
3.11 Detailed Procedure of Forming a Wi-Fi Direct Network 28
3.12 Bluetooth Scatternet 34
3.13 Classic Bluetooth Protocol Stack 35
3.14 BLE Protocol Stack 36
3.15 CoAP Abstract Layering 38
3.16 CoAP Message Format 39
3.17 Procedure for Saving and Finding data in DHT 41
3.18 Consistent Hashing Illustration 42
3.19 Node Addition and Removal 42
3.20 Consistent Hashing Ring Balanced by Virtual Nodes 43

4.1 System Architecture 45
4.2 Requests and Responses Size 48
4.3 The organization of Tuples and Different Level of Tuple Spaces 49
4.4 Topology of the Tuple Spaces 50
4.5 Example of Engagement 51
4.6 Example of Replica Mechanism 53

5.1 Architecture of the Coordination Framework 54
5.2 Relation between Layers 56
5.3 Implementation of the Consistent Hashing 57
5.4 Flowchart of the Replica Management Component 60
5.5 Replica Mapping 61
5.6 Disengagement Example 62
5.7 Organization of Message and Service Layer 63
5.8 Organization of the CoAP Implementation 65
5.9 Key Member Functions of FrameworkCore 69

6.1 Wi-Fi Direct Data Rate 71
6.2 BLE Data Rate 71
6.3 Throughput Between 2 nodes 72
6.4 Packages per Second Between 2 nodes 72
6.5 Throughput of the Coordination Framework (2 Nodes in the System) 73

6.6 Packages per Second of the Coordination Framework (2 Nodes in the System) 73
6.7 Throughput of the Coordination Framework (7 Nodes in the System) 74
6.8 Packages per Second of the Coordination Framework (7 Nodes in the System) 74
6.9 The Performance Ratio of the Coordination Framework 74
6.10 Average Time-cost for Engagement and Disengagement 75
6.11 Hit Rate During Engagement and Disengagement 75
6.12 Data Distribution of the System 76

Chapter 1

Introduction

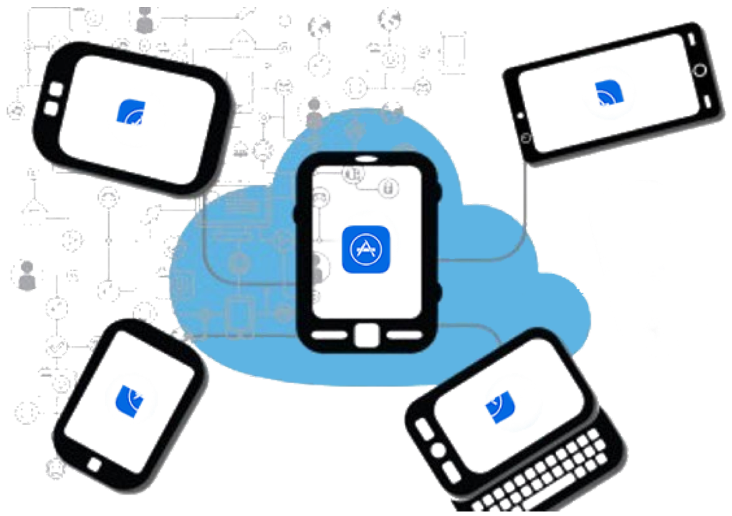


Figure 1.1: Multiple Devices Work Together for a Single Application

Similar to multi-thread programming or multi-CPU programming, where multiple functions running on different threads or CPU can work together to significantly improve the performance of a single application, multiple devices within a local network should be able to work together to enhance the performance and capacity of a single application. Improving the performance and capacity of a single application means that devices can share not only the CPU but also the memory, disk space, and other hardwares such as screen and sensors. Moreover, because a distributed mobile application combines all the devices together, making the application look like it is running on a single virtual machine, it should be able to simplify the development of distributed applications significantly. Figure 1.1 above, demonstrates an example of five devices working together to operate a single application. Each of the devices, with the exception of the center device, is responsible for a portion of the application. The center device, referred to as the coordinator, coordinates all other devices. The key component for implementing such infrastructure is called the coordination model, which takes charge of coordinating nodes in the cloud so that they can work together properly. There are many well-known coordination models for distributed applications running in a large data center. However, there is only a limited number of coordination models specifically designed for a mobile environment [1, 2, 3].

Such a problem does not attract much concern until mobile devices start playing increasingly important

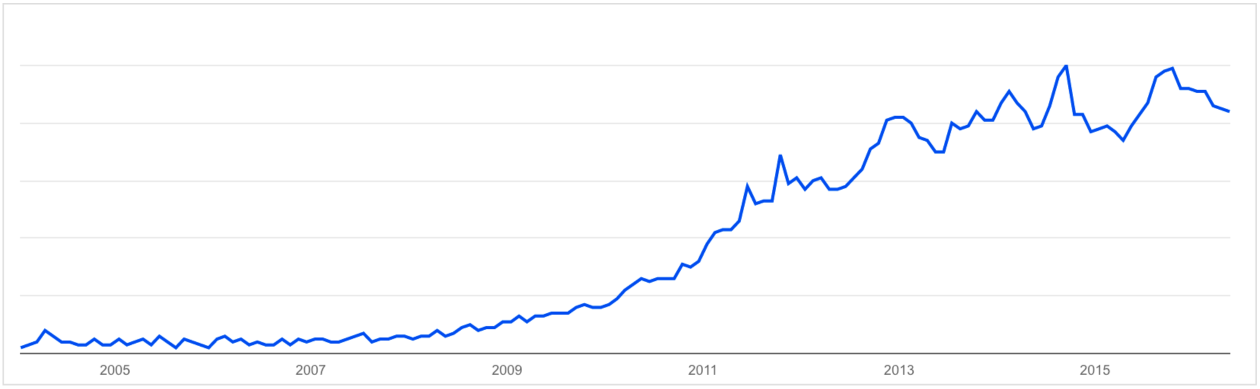


Figure 1.2: Search Interests Trends on Mobile Cloud Computing[4]

roles in people's daily life. According to a survey about mobile devices from IDG [5], the population of individuals who own at least one tablet has tripled from 2011 to 2014. Another survey [6] offered by Sophos Labs reveals that people in North America and Europe are now carrying, on average, 2.9 devices with them every day. More mobile devices means that developing cloud solutions for the mobile environment is beginning to receive more attention. In fact, as figure 1.2 shows, the key word "mobile cloud computing" on the Google platform has grown rapidly since late 2010 [4].

On the other hand, with the help of 3G networks and new phones with powerful OS and advanced wireless network capabilities, the smartphone has gained its popularity all around the world. A report done by KPCB shows that by the end of 2015, the number of global smartphone users has reached over 2,500 million [7]. According to the research done by Morgan Stanley in late 2009, people believe that the population of mobile users will overtake desktop users in 2015 [8]; another report by comScore published in early 2015 proved the prediction [9] as well. The popularity of smartphones increased the pace of the development of technologies behind it, such as operating systems, wireless communication technologies, and mobile distributed systems - all fundamental technologies of mobile cloud computing.

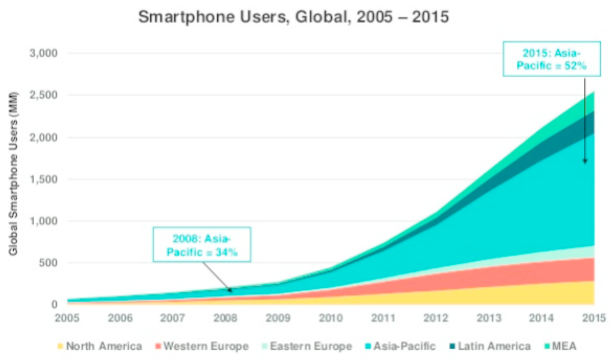


Figure 1.3: KPCB Internet Trends 2016[7]

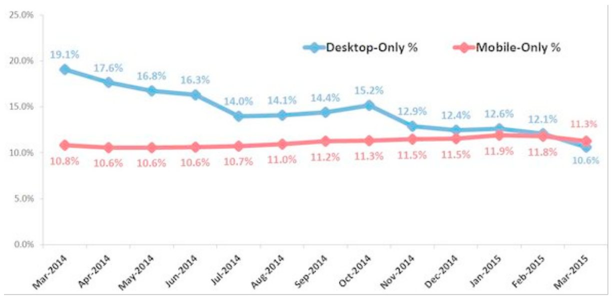


Figure 1.4: Single Platform Users' Share[9]

Along with the proliferation of mobile devices, the software running on the devices is consuming even more hardware resource than before. At the beginning of the mobile computing era, the major uses of the smart devices were mostly just simple tasks like calendar, email, and texting. Therefore, mobile devices were always equipped with low-end hardware. For example, the Nokia 9000 communicator, manufactured in 1996, came with an Intel 24 MHz CPU and 8MB of memory [10]. The hardware limitations of these early devices became apparent as people began using their mobile devices in more scenarios of their daily life, particularly with greater numbers of people playing games on their smartphones and tablets. When running the popular and resource-intensive game Need for Speed, for example, the FPS (frames per second) will drop to under 30 when rendering complex objects such as a car crash. The drop in FPS occurs even if the game is running on a powerful tablet equipped with dual core 2GHz CPU and 4GB RAM. Since many people now carry more than one device with them, and because a person can only focus on using one device at a time, it would be ideal if the idle hardware resources could be utilized to help to calculate the render result. That being said, it is of great importance that a software developing technique enabling multiple nodes to be distributed on different devices to work together for a single application.

Besides the aforementioned gaming scenario, the demand of multiple devices working together is critical in medicine and business areas as well [11]. Take the following health care solution for example. I was helping a company design a health care solution that can continuously monitor a user's health status by gathering and analyzing the information from several different sensors placed on the user's body. Detecting health issues needs to take multiple information into synthetic consideration. For example, to detect the emotional state of a user, the application needs to consider not only his heart rate but also information from the gyroscope and accelerometer to obtain data regarding his movement. Only when a user is encountering both high heart rate and constantly moving back and forth can his emotional state be agitated. Therefore, a technique framework that can coordinate all the sensors and devices working together properly is needed. A business-related example of multiple devices working together is an Internet of Things (IoT) project that I worked with a company in Ontario. The company had developed a smart device that was equipped with many sensors to monitor information like temperature, air humidity, moisture and PH value of the soil. The device was developed so that farmers could gather data from multiple locations on their property by placing individual devices. All of the devices would work together as a system to provide better strategies for farm and crop management. P2P technology was a better option when designing the system because client/server architecture is not robust enough to support the intended use. Therefore, generating strategies such as irrigation timing and fertilizing need to involve multiple devices.

The scenarios also present other challenges to the coordination model. The first challenge is the processing efficiency. In the health care solution example, it was extremely important to coordinate all of the involved sensors and devices in real time. Inefficient detection and reaction might have caused serious consequences especially when an emergency situation occurred. For example, if the system failed to detect a heart attack of a user in time, or if it could not perform an instant reaction such as calling a doctor, the system might cause the user's death. Therefore, efficiency is the highest priority when designing a coordination model.

Furthermore, on account that in most cases it is difficult to establish stable connections between nodes under the mobile environment, network connectivity is another important issue that needs serious consideration. The traditional client/server architecture is no longer a good choice for the mobile environment due to the limitation of maintaining stable connections, therefore decentralized P2P (peer to peer) technology seems to be a better solution. In the example of the IoT project, most farms are located in rural areas without internet access, and it is impossible to have a central server coordinating all of the distributed devices. Therefore, the devices have to form a robust P2P local network by themselves to communicate with each other. Establishing a P2P network is not an issue because there are many sophisticated P2P solutions for either hardware or software [12]. The real challenge is how to make the P2P network robust enough in order to sustain the applications running on it. A weak P2P network might cost high package loss rate and slow data routing speed, which will significantly impact the performance of the system. Such an impact is certainly unacceptable for most business scenarios. Thus developing a robust self-organized P2P network is essential for designing a new coordination model.

Developing a self-organized P2P network is not the only important aspect of a robust P2P network. Scalability is also one of the most important aspects of a robust P2P network. Scalability means the distributed P2P system itself should be able to properly handle the expansion and shrinkage of the system. Because P2P systems are always decentralized, disengagement of a single node should not impact the whole system, and it should be able to welcome newly engaged nodes at any time. For example, if a farmer enlarged his farm and placed more smart-devices on it, the system should be able to accept the new devices and automatically involve them in working together. On the other hand, if a farmer needed to remove or replace some of the devices, the system should also be self-adaptive. Scalability implies the requirement of fault-tolerance within the coordination model because encountering failure status can be one of the primary reasons causing the disengagement of a device.

Ultimately, people's increasing dependence on the mobile environment requires us to develop a better coordination model that addresses the demand of the mobile scenario. Some coordination models have been designed and implemented for the mobile environment, and LIME (Linda In a Mobile Environment) is one of the most well-known coordination models for mobile scenarios. It is the extension of Linda, which is recognized as the earliest coordination model, and it introduces many important features to facilitate the development of mobile applications [3]. However, many of LIME's drawbacks have prevented it from being widely deployed. This paper introduces a new coordination model that uses the consistent hashing and replica mechanism to obtain better performance, which keeps the advantages of LIME intact while improving upon its drawbacks. Moreover, based on the new coordination model, this paper also presents a cross-platform coordination framework utilizing several technologies such as CoAP protocol, P2P networking, Wi-Fi Direct, and Bluetooth Low Energy. The coordination framework is designed to simplify and speed up the development of distributed mobile applications.

The remaining parts of the paper are constructed as follows:

- **Chapter 2 - Problem Definition** discusses what problems need to be solved by this research.

- **Chapter 3 - Literature Review** reviews two previous coordination models and relevant technologies for developing the new coordination model and framework.
- **Chapter 4 - Solution and Architecture** illustrates the design of the new coordination model.
- **Chapter 5 - Implementation** depicts the detailed implementation of the coordination framework.
- **Chapter 6 - Experiment** tests the performance of the coordination model and framework under different scenarios.
- **Chapter 7 - Conclusion and Future Work** describes this research's next steps.

Chapter 2

Problem Definition

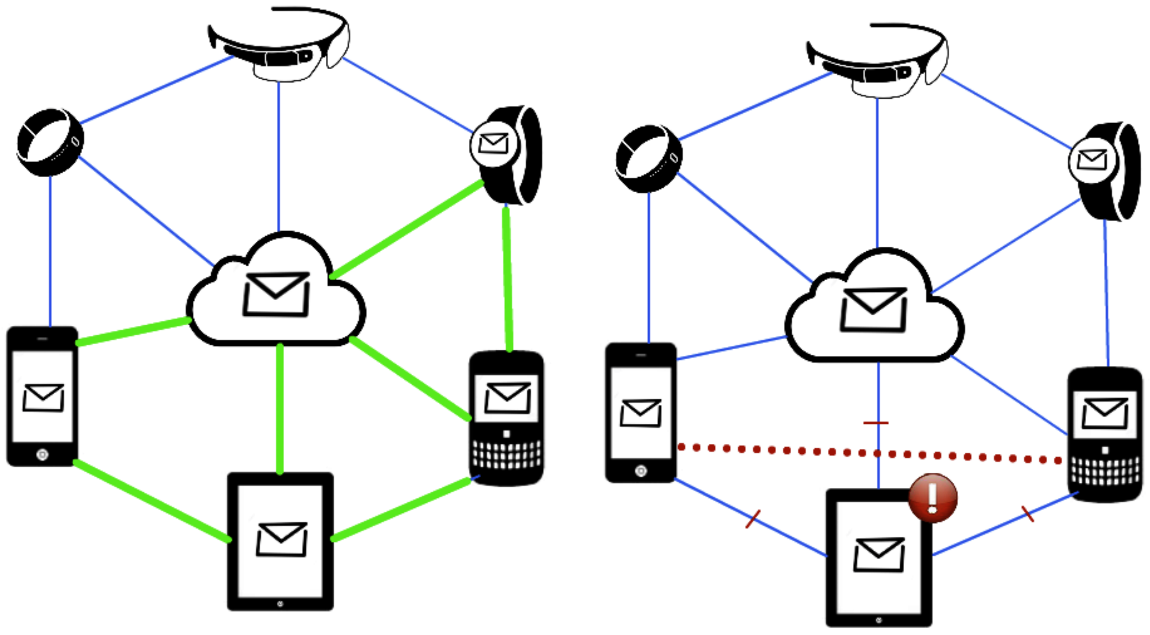


Figure 2.1: Coordinations of Distributed Mobile Applications

People are now carrying more devices with them on a daily basic and relying on them more than ever before. A coordination model specifically designed for the mobile environment is in demand. After reviewing the most dominant coordination models, which will be detailed in the next chapter, this research realizes that the most important requirements of developing a coordination model are:

- High-performance

  A high-performance coordination model means it can efficiently coordinate all of the operations. The coordination model itself must not notably affect the throughput of the system.

- Fault-tolerance

  Fault-tolerance requires the coordination model to guarantee both the system availability and data integrity when errors occur. Guaranteeing the system availability means when some nodes are encountering errors, the system still needs to function normally. Data integrity means that no data will be lost during the engagement and disengagement procedure.

The left side of figure 2.1 gives an simple example of a high-performance coordination model. In this example, once a device generates a new message, all related devices should be able to instantly receive the message. The right side of figure 2.1 explains the definition of fault-tolerance. If the tablet, depicted as the bottom device, accidentally leaves the system, then the system can still function normally, without losing any of the messages by cutting off connections that are related to the tablet and reestablish a connection between the two smartphones.

There are several foreseeable challenges in developing a high-performance and fault-tolerant coordination model.

- **P2P Communication Technology**

  When dealing with mobility, having a central server in the system might become impractical in many cases. Therefore, the coordination model must use P2P technologies to perform the networking. There are many mature wireless P2P technologies for mobile devices to communicate with each other like Wi-Fi, Wi-Fi Direct, Bluetooth (low energy), or NFC. However, since each technology has its advantages and disadvantages, finding a solution that uses one or more above technologies with the best trade-off between system feasibility and data transmitting performance could be an immediate challenge.

- **Fast and Scalable Data/Resource Locating Algorithm**

  The requirement for efficiency and performance of the coordination model demands a fast and scalable data/resource locating algorithm. This research needs to find a way that allows users to rapidly and precisely locate the required data or resource, especially when there is a large amount of data/resources in use. There are many search algorithms such as linked lists, binary search, and hash table. However, these search algorithms are all designed for in-memory computation running on a single machine. This means that these algorithms will no longer be available when multiple devices are involved. Therefore, a search algorithm that supports distributed searching must be developed. Besides searching speed, scalability is yet another requirement needed for the search algorithm. The arrival and departure of nodes might occur frequently and some unexpected situations might force the data to be rebalanced to other nodes in the same mobile cloud. The search algorithm is required to be self-adaptive to these changes.

- **Engagement and Disengagement of Nodes**

  The coordination model should be able to handle the engagement and disengagement of nodes. This requirement includes two aspects:

  1. How to discover existing nodes and services, and how to detect the departure of leaving nodes.

2. How to properly handle the arrivals and departures without impacting the system's performance.

- **Data Balancing**

  When the amount of data expands explosively, or new nodes join the system, to maximize the system's capability and obtain better performance, the coordination model is required to keep the system balanced. Therefore, how to migrate data between nodes without jamming the network traffic is yet another problem lying in front of the coordination model.

- **Fault Tolerance and Data Integrity**

  While errors occur in some of the nodes in the cluster, or flood requests are routed to a small group of nodes, some nodes might enter the failure state and become offline. Moreover, disengagement might cause the data and resources on the leaving become offline as well. The coordination model must guarantee that no data will be lost and the system can still function properly when these situations occur.

Chapter 3

Literature Review

The following sections detail the research undertaken for designing the coordination model and framework. This includes research in evaluating current coordination models and finding solutions to the challenges described in the Problem Definition chapter.

3.1 Coordination Models and Languages

According to the definition provided in a paper by Carriero and Gelernter, "a coordination model is the glue that binds separate activities into an ensemble" [13]. That is to say, a coordination model needs to provide a solution to properly coordinate all the independent units to work together as a single virtual ensemble. A typical coordination model should consist of three parts [2]:

1. Coordination entities

   Coordination entities are the actual units that being coordinated. A coordination entity can be a process, a thread, a concurrent object or even a single device.

2. Coordination media

   Coordination media are the media that carry the communication between entities. For example, threads synchronization of Linux system can be considered a classic coordination model where the threads are the coordination entities and the synchronizing mechanism, like semaphores and mutex locks, are the coordination media.

3. Coordination laws

   A coordination model should define several rules for the coordination entities and media to follow. These rules describe how coordination entities being coordinated through the coordination media by using a number of coordination primitives. Such rules are called coordination laws.

A coordination model can be defined by a coordination language. Each coordination language will define a unique coordination model. Coordination models were designed to be responsible for the decoupled interactions and cooperations among concurrent processes [3]. There are two major categories of coordination model: *data-driven* and *process-oriented*

The main characteristic of the data-driven coordination models is that, although a data-driven coordination model may also be able to coordinate the computational components (e.g. the processes or threads) of a system, it is more inclined to coordinate the values of the data being received or sent. Contrary to the data-driven coordination model, a process-oriented coordination model can only coordinate the computational components. The actual values of the data being manipulated by the computational components are never involved in the coordination procedure of a process-oriented model [3]. As shown in the examples of the IoT project and the healthcare project described in the Introduction chapter, the actual values of the data are the essential elements being coordinated. Therefore, this study focuses on the data-driven coordination model only.

Many data-driven coordination models have been invented among which LIME is the most appropriate one that provides host-to-host coordination in mobile ad-hoc networks. The coordination method of LIME is inspired by a previous coordination model defined in a coordination language called Linda [14].

3.1.1 Linda

Linda was first introduced in the paper "Linda and Friends" in 1986 and it is considered the earliest coordination language [3]. Linda was invented to solve the challenges of parallel programming and the main mission for it was to coordinate the parallel processes and threads running on a multi-CPU computer [15].

According to the author, dealing with spatial problems and temporal relationships among parallel processes are the most complicated issues for parallel programmers. Linda solves them by decoupling the cooperation between concurrent processes in space and time [15, 3]. In parallel programs, the output generated by a process will always be accepted by certain other processes as the input. Traditionally, sender processes need to know exactly how to locate the target processes and how to communicate with them. Data managing strategies such as whether the generated result will be used in the future or not are also affecting the processes' behaviour. Thus, addressing the aforementioned issues has always been time-intensive for parallel programmers. Decoupling in space means the sender process is not required to know about who the receivers are and how to properly deliver the data to them. Instead, the sender only needs to tag its new data with a logical label such as a name suffixed with the sender process ID and then store the labeled data into a virtual space called tuple space. As indicated by the naming of tuple space, such labeled data are defined as tuples. Once the sender has finished storing the tuple into tuple space, the sender process does not need to care about the tuple anymore. When one or more processes wants to access the tuple, they can just directly turn to the tuple space and retrieve it.

Temporal relationship means the relationship between processes exists only when they need to communicate with each other [13, 15]. Apparently, such a relationship is ephemeral and both the sender and receiver processes need to be presented and ready simultaneously. However, such requirement is undesirable in some situations. For example, in a heavy load system, a sender might generate a large number of messages pending to be processed by multiple receivers, but it will not be able to send the next message until the receivers all accept the current message because of the constraint of temporal relationship. Accordingly, the data stream processing speed has been vastly limited. Similar to decoupling in space, decoupling in time means that the receiver does not need to be present when the sender sends a message. Linda accomplishes this by occupying a shared memory space which is independent of all other processes involved in the parallel system. This shared memory space is used as the tuple space to store all the messages, i.e., tuples.

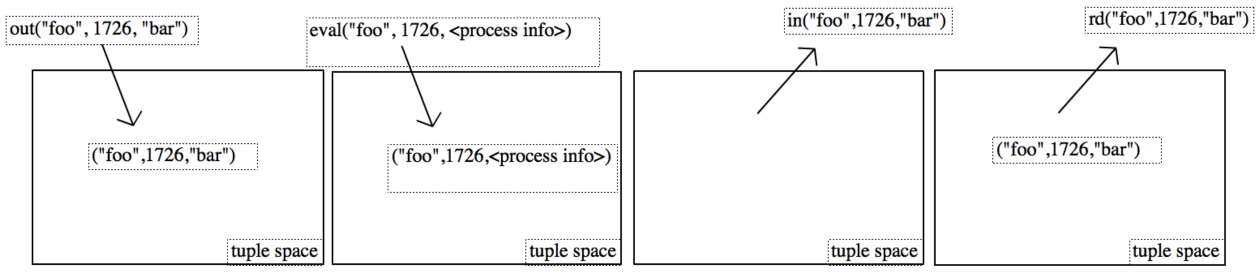


Figure 3.1: Fundamental Operations in Linda

As mentioned above, unlike conventional memory management whose storage unit is the physical byte, Linda uses the logical tuple to be its memory unit. Where the elements in conventional memory are accessed by address, elements in Linda are anonymous and accessed by means of associative pattern matching. In Linda, tuples are sequences of typed parameters which can be divided into two different types: *actuals* or *formals*. *Actuals* are the actual value of the tuple. For example, the first 3 items of a tuple (“foo”, 1726, “bar”, ?int, ?string) are the actuals which mean the label of the tuple is “foo”, it comes from process 1726 and it contains a related value “bar”. *Formals* are areas that indicate the type of related fields. In the previous example, the last two parameters are the formals suggesting the seconding parameter should be an integer and the third one should be a string. Formals are like “wild cards”. They are used to match against the actuals during the pattern matching to select one or more tuples from the tuple space. Actual is required for a tuple but formal is optional. While accessing a tuple without formals, only one tuple whose actuals exactly matches the searching pattern will be returned.

Linda introduces four fundamental operations on the tuples, which are: *out(t)*, *eval(t)*, *in(t)* and *rd(t)*. Out(t) operation will save a tuple t into the tuple space. For the example in figure 3.1, *out(“foo”, 1726, “bar”)* means save a tuple named “foo” from process 1726 to the tuple space and its value is “bar”. eval(t) operation is similar to out(t), but instead of saving a simple string value, it actually saves an active process to the tuple space so that its execution can be continued later on or by other nodes. Similar to the concepts of multitasking and interrupt handling in the operating system, eval(t) provides a way for context switch in Linda. In(t) operation retrieves a tuple from the tuple space and the execution is finished, it will remove the original copy of the tuple from the tuple space. Rd(t) will also retrieve a tuple from the tuple space, but unlike in(t), Rd(t) will leave the original copy in the tuple space.

3.1.2 LIME - Linda In a Mobile Environment

Linda decouples the communication between nodes both in time and space, which means it resolves individual processes from taking care of the complicated network communication issues and coordinating actions. This decoupling feature is appealing and ideal for the mobile environment, as dealing with network issues such as unpredictable engagement/disengagement of new nodes and routing the requests among the distributed mobile devices are the most critical challenges for coordination in the mobile environment [16]. Additionally, the idea of Linda has been widely accepted by many scientific research communities ranging from computer science to economics [17], and has been adopted as the core for investigating technologies and methodologies interested in coordination of nodes in complex systems. Therefore, researchers chose Linda to be the foundation and tried to adapt it to the mobile environment.

Linda was designed to coordinate processes and threads of a parallel system running on a single machine. The first challenge the researchers experienced was how to make Linda compatible with running on multiple machines. Early attempts on extending Linda to the distributed environment all relied on stable and long endured connections, and assumed disconnection would only happen accidentally [16, 14]. Such attempts certainly simplified the implementation. However, this was not enough for the mobile environment. In the mobile environment, disconnection may happen oftenly due to unpredictable actions. For example, the connection will be lost when a user carries one of his devices connected to the mobile cloud out of the connection range or when the user terminates the process that is performing the connection. The connection can also be cut off by switching the cellular base station or by the system itself for battery conservation purposes. Moreover, the idea of global tuple space might not be applicable in the mobile environment since global tuple space requires stable connections. In short, mobility demands fewer constraints on the network and more flexible reconfiguration of its contents [18].

To overcome the difficulties, researchers developed a new coordination model called LIME, which is the acronym for Linda In Mobile Environment [16, 14]. In LIME, mobile devices are defined as mobile hosts and the processes being coordinated are called the mobile agents. LIME sees mobility in two aspects: physical and logical. Physical mobility is the actual movement of the mobile hosts in physical space, such as roaming from network to network. Logical mobility allows mobile agents to travel among different mobile hosts. Instead of the global tuple space in Linda, LIME introduces a new notion called *transiently shared tuple space* to tie the physical and logical mobility together.

Figure 3.2 illustrates the organization of the transiently shared tuple space in LIME. Each mobile agent is associated with one or more *interface tuple spaces* (ITS) which will store all the tuples shared by the agent. An ITS will be bound to a mobile agent permanently and will be transferred with the agent together when physical movement occurs. Tuples can be accessed through the same Linda operations *in*, *rd*, *out*, and, *eval* as described in previous sections. Tuples are still anonymous and need to be selected via pattern matching. However, mobile agents, hosts, and ITS need to be named. Additionally, each mobile agent and host need to be given a unique identifier in the global co-located environment. Dot notation is used to identify which

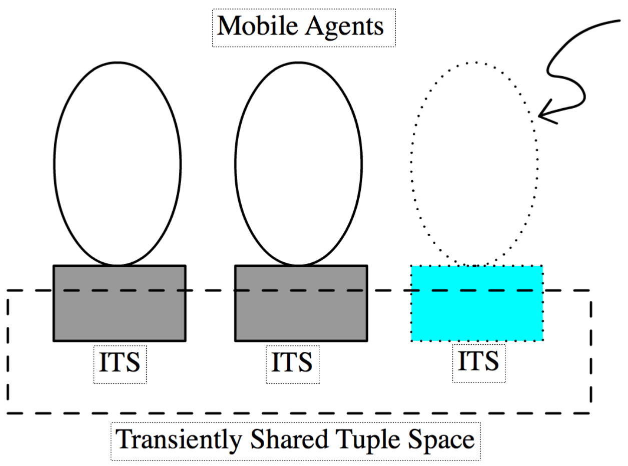


Figure 3.2: Transiently Shared Tuple Space in LIME

ITS is performing the specified operation, e.g., *x.in(t)* means an ITS named x is performing operation *in* on tuple t. A mobile agent can hold multiple ITSs, and the ITSs can be configured public or private. Public ITSs are subject to sharing while private ITSs can only be accessed by the agent itself. ITSs are shared independently with the corresponding ITSs in other co-located agents. For example, agent A owns ITSs a, b, and c, while agent B owns ITSs b, c, and d, then only b and c will be transiently shared between agent A and B. By "transient" it means the relationship between agents is not permanent and will be changed dynamically due to mobility. Moreover, the contents in the tuple space of LIME will change according to the migration of agents as well. All of the shared ITSs are virtually merged and exposed to agents that are currently connected. In other words, if agents A and B both have a shared ITS named x and they are currently connected in a local network, once A finished performing operation *x.out(t)*, tuple t is available for agent B to retrieve immediately.

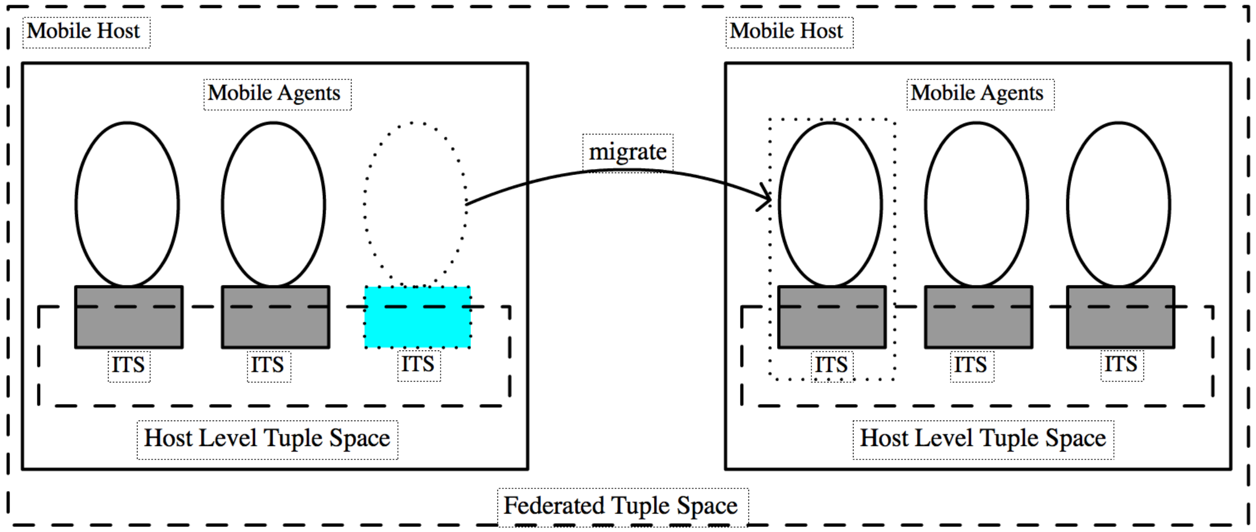


Figure 3.3: Topology of the Tuple Space in LIME

Figure 3.3 depicts the design of the tuple space of LIME in more detail. Physical connectivity is performed by the mobile hosts. Agents are logically bound to hosts and are allowed to migrate from one host to another. Apparently, agents co-located on the same host are mutually connected. Therefore, all of the agents' ITSs can be combined to form a host level tuple space in which logical mobility will occur. All the connected hosts merge their host level tuple spaces into the federated tuple space where physical mobility happens.

LIME defines a special ITS called LimeSystem to store all the necessary runtime configuration - in other words, the topology of all current ITSs and the information about currently involved agents. LIME requires the LimeSystem ITS to be accessible by any co-located agent at any time, and only the system itself can modify the content in it, meaning the content is read-only to all of the agents.

LIME makes the content of an ITS accessible to other by continually reconfiguring the set of co-located agents. When a new mobile host is connected, the agents resting on it will be involved and the current tuple space will be completely recomputed by taking the ITSs of the new agents into consideration. Such procedure is called the *engagement*, and it will guarantee every shared ITS is accessible to corresponding agents once it is finished. Similarly, re-computation will also be applied to guarantee the data integrity and consistency when *disengagement* happens, i.e. the departure of mobile hosts and agents.

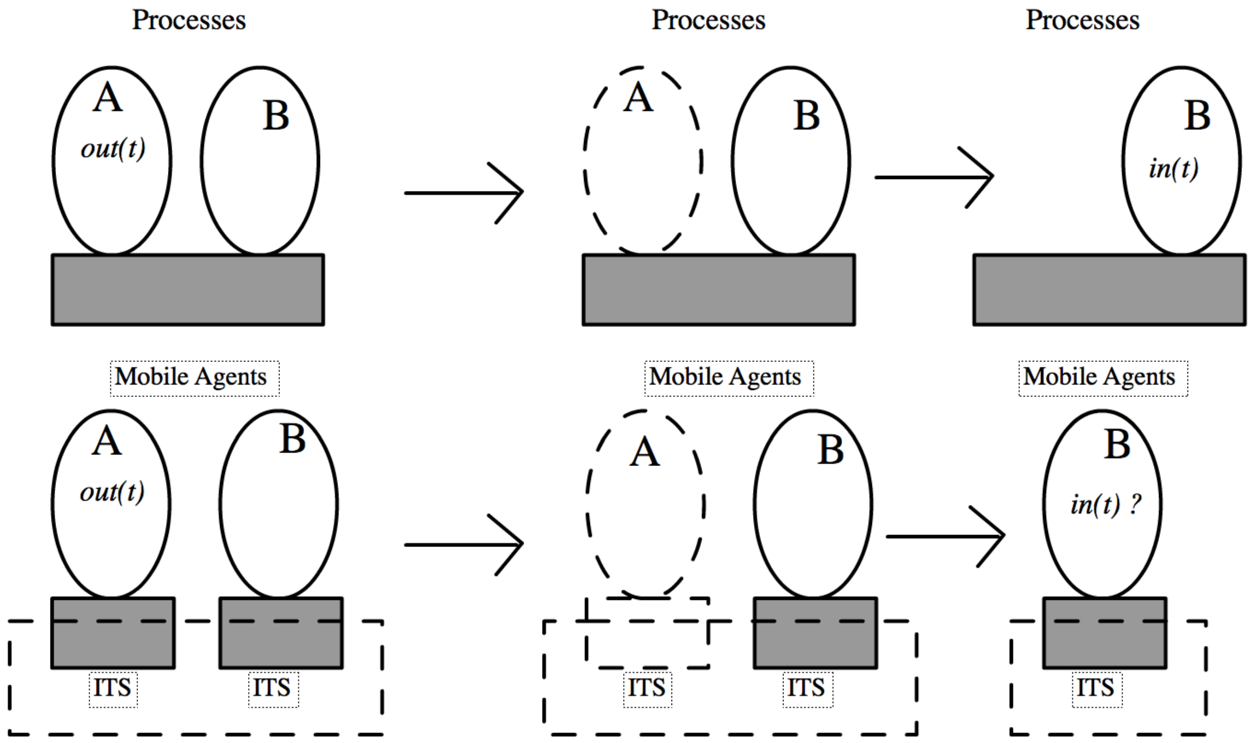


Figure 3.4: Tuple Space in Linda vs. Transiently Shared Tuple Space in LIME

Keeping the feature that communication between agents should be decoupled in time and space becomes a major challenge to LIME because of the transient nature of its shared tuple space. LIME introduces the concept of location-ware computing to master it. Figure 3.4 describes the typical scenario of this challenge.

Suppose we have two nodes, A and B, that are being coordinated. If A leaves some data that is needed by B and then terminates itself for whatever reason, how do we make sure that the data is still available even when A has been terminated? The solution is naturally solved by Linda, which is depicted in the upper row of Figure 3.4. In Linda, process A packs the information needed by B as a tuple t and saves the tuple t to the tuple space by executing an *out(t)* operation (step 1). Because tuple space in Linda is permanent and globally accessible, the tuple t is able to remain in the tuple space even if its producer A has already gone (step 2). Eventually, B can obtain the information stored in the tuple t by running *in(t)* whenever it needs (step 3). However, things become harder in LIME, as depicted in the lower row of Figure 3.4. After agent A has stored a tuple t to its ITS by executing *out(t)* operation, tuple t becomes visible to B immediately thanks to the transient sharing mechanism (step 1). Unfortunately, all the ITSs of agent A have been taken away along with A's departure (step 2). Thus, when B attempts to retrieve tuple t by performing *in(t)* operation after A has already disappeared, B will be blocked since no matching for tuple t can be found.

This problem is caused by the fact that there is no persistent tuple space in LIME. Mobile hosts in LIME are all temporarily connected, so the content of the federated tuple space varies dynamically. LIME solves the problem by explicitly specifying the location of the tuple when performing operations. LIME adds a new parameter to annotate the destination of the tuple being operated. When executing saving operations like *out* or *eval*, a copy will be saved locally in the destination agent's ITS if the destination is different from the original agent who launched the operation. LIME requires all its implementation to provide runtime support for managing the location information. The implementation needs to examine the whole system either passively or actively and detects the misplaced tuples in time. Once the system detected tuples whose intended destination is different from their current location, namely, "misplaced" tuples, it will try to move them to their correct places. For instance, in step 1 of the previous example, A will run *out[B](t)* instead of simply running *out(t)*. Two actions will be taken after A launched the request. First, t will be stored in A's local ITS. Second, the system notices that the destination B is different from the current location A. Therefore it will check whether B is currently connected to the system, if yes, a copy of t will be stored in B's corresponding ITS. Otherwise, the system will try again once B is ready. LIME also requires combining the two actions together as a single atomic operation to secure data integrity.

Such design implies that tuples are re-partitioned during both engagement and disengagement. Whenever a new host is connected to the system, LIME will go through all ITSs in the current system to check whether their destination matches the agents in the new host or not. If yes, then LIME will place a copy of the matched tuple to the ITS of the new connected agent. Similar to the engagement, LIME will go over the federated tuple space to remove all the tuples whose desired destination is the leaving agent when an agent leaves the system.

Furthermore, for retrieving operations such as in and rd, LIME allows the user to use not only the destination (represented as λ) as the parameter but also the current location (represented as ω) for selecting tuples. Take in operation for example, *in[ω, _](t)* limits the operating scope to the tuples who came from ω;

in[$\_$, λ](t) means only tuples whose final destination is λ can participate in searching for tuple t; *in[ω,λ](t)* restricts the scope to the tuples who came from ω and going to λ.

In a rapidly changing environment, it is impossible for a system to automatically provide a perfect solution for handling every event such as connection/disconnection, new contents available and changing of the topology. Therefore, the system should be able to expose some interfaces to the programmer to enable them to specify how to properly react to events. LIME introduces the notion of *reactsTo* to accomplish this.

Events are represented as tuples in LIME and will be stored in the LimeSystem ITS which provides runtime support that continually monitors the status of the current system. Events in LIME consists of the arrival and departure of agents, connectivity changes, and topology changes such as agents' migration. When one of the events happens, LIME system will transform it to an event tuple and insert it to the LimeSystem ITS. In order to enable the programmer to deal with the events, LIME introduces the *react* operation under the form *T.reactsTo(s, p)* where T represents an ITS and s is a code fragment provided by the programmer to be executed when an event tuple matching the pattern p is found in T. All the reactive statements will be registered to the LimeSystem ITS as well. Programmers can also specify the location parameters for a reactive statement when registering it, e.g., *T.reactsTo[ω, λ](s,p).* Once LIME system detects the occurrence of an event, it will retrieve all the registered reactive statements that are declared to handle the event and then trigger them one after one.

In sum, LIME adapts the Linda model of coordination to mobile environment by involving the ideas of transiently shared tuple spaces, location-aware computing, and reactive statements. LIME enables rapid and dependable development of distributed applications running in the mobile environment [19].

3.2 Peer-to-Peer Network

As explained in the previous sections, centralized architectures like the classic client/server architecture are not suitable for the mobile environment. Therefore, peer-to-peer (P2P) must be supported by coordination models targeting mobile environment.

3.2.1 P2P Brief Introduction

P2P is a distributed programming architecture that partitions missions between peers and provides the communication fundamentals for peers to work together [20]. In a P2P network, peers can be anonymous and are participating under equal privilege to make a portion of their resources available to other peers without requiring a central server to coordinate the requests [21]. According to the definition provided by researchers at the Technical University of Munich, only when a network architecture meet the following conditions can it be called a P2P network: [20].

- All participants of the distributed network architecture share a part of their hardware resources (e.g. computing power like CPU or GPU, memory, disk storage, screens, printers) which are necessary to perform the service or generate the content of the network;

- All participants can directly access the resources without any intermediary entities;
- Participants are all equivalent in privilege, which means participants in such network are both resources providers and resources consumers;

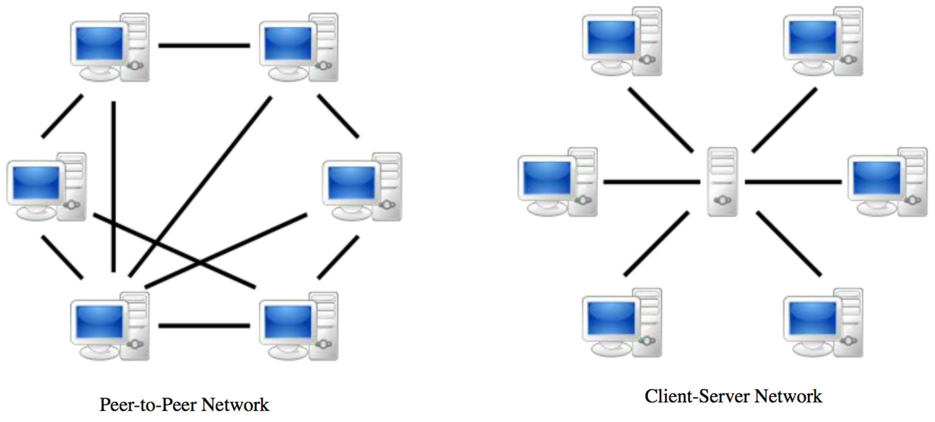


Figure 3.5: Examples of P2P network and classic Client-Server Network

The left side of figure 3.5 shows a typical example of a P2P network in which peers are mutually connected and share resources with each other without using a centralized server. It also points out that the requirement of equal privilege does not mean every peer needs to be exactly the same. Peers can share different resources with each other. "Equal privilege" means there must not be any privileged "super peer" in the network. The departure of any nodes should not seriously impact the service quality of the P2P system. The other part of figure 3.5 contains an example of traditional client-server architecture where all the services are provided by centralized servers. Centralized servers are privileged nodes which the whole system relies on. In contrast to the P2P system, resources providers and consumers are divided, meaning that centralized servers are responsible for providing resources for the individual clients to consume. Additionally, offline servers might cause the entire system to be inaccessible. Most distributed systems within the mobile environment are built on a foundation of unstable networks and are required to be self-organized, hence, the client/server architecture is not acceptable for most mobile scenarios. Besides decentralization, another feature of P2P network architecture is that it essentially supports scaling out the system by adding more peers to achieve higher performance [22]. On account of the fact that scaling up a mobile device is impossible in most cases, such scaling out feature of P2P seems to be ideal for developing a mobile cloud solution.

3.2.2 History of P2P

The idea of P2P had been presented many years ago and similar ideas had already been used in many applications [23]. ARPANET, the precursor of the internet, was designed for client-server architecture but its idea was that every node in the network can request and serve content implicitly, which described the key concept of P2P. However, due to the lack of ability to provide a context-based routing mechanism and self-organization, which are also the indispensable components of P2P network, ARPANET cannot be considered the first P2P implementation [24].

Later on, following the rules of P2P, a distributed messaging system called USENET was invented in 1979 [25]. USENET was a distributed discussion system that allowed users to publish articles to different categories. Categories were served by a large and constantly changing cluster of servers where no central server or administrator was present, which meant USENET was a decentralized architecture and it was extremely hard to apply censorship and restriction on it. USENET defined several rules for managing the messages among the network and guarantee the self-organization of servers. Therefore, USENET can be considered the early form of the P2P network [26].

P2P architecture was not popular until the development of Napster [27, 24], a distributed music and file sharing application that ran on the internet. Napster established a virtual network among users that was totally independent of the physical network and allowed the users to share music and files without any censorship or restriction. Such an idea was adopted by BitTorrent, and soon afterwards P2P architecture was popularized all over the world [28].

3.2.3 Taxonomies of P2P architecture

A typical P2P network is required to implement a virtual network over the physical network. Data is still exchanged directly under TCP/IP protocol at the physical layer but communication between peers will not be held directly by the physical layer. Instead, the virtual network will perform all the communication and allow peers to connect to each other directly via logical overlay links. Such mechanism enables the P2P system to be entirely independent from the physical network topology. Based on how resources are identified and how the peers are linked to each other in the virtual network, P2P networks can be classified as *unstructured* or *structured* [29].

Unstructured P2P network

Figure 3.6 depicts an example of an unstructured P2P network. In an unstructured P2P network, peers randomly connect to each other without imposing a particular structure [30]. Since there is no requirement applied to the peers to form a designated structure upon the virtual network, it is easy to build an unstructured P2P network and perform partial optimizations to different regions of the overlay links [31]. Moreover, unstructured P2P network is a naturally robust network because all the peers in the network share the same role which means frequent arrival and departure of nodes will not significantly affect the whole system [32].

However, lacking of structure presents significant drawbacks to an unstructured network. If a node wants to select a specific piece of data in an unstructured P2P network, the only way to do so is to broadcast a request to every node in the network, and nodes that contain the desired data will send the data back to the

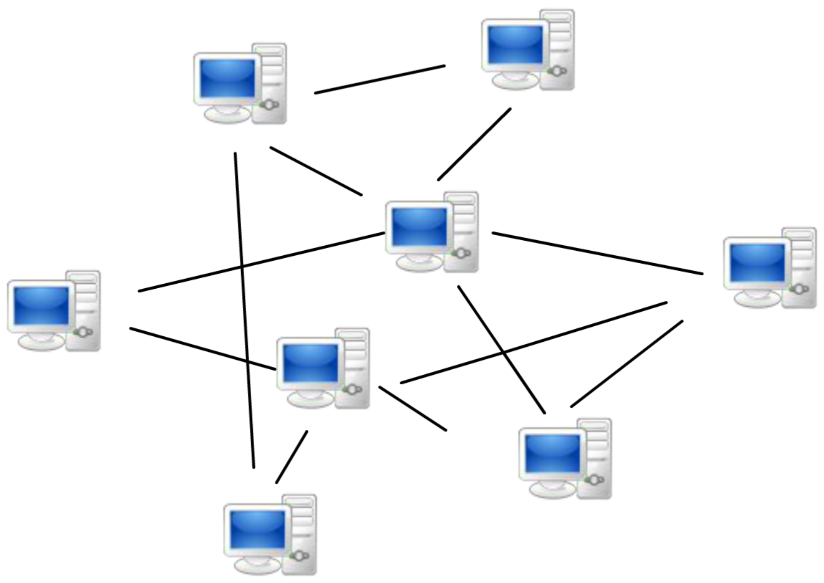


Figure 3.6: An Example of An Unstructured P2P Network

original node. This will cause a flood to the system which will jam the network traffic and consume more CPU time and memory space [33].

Structured P2P network

Unlike the unstructured P2P network, peers are organized according to a specific topology in a structured P2P network. The particular structure of the network ensures the efficiency for finding resources in the network [34].

Figure 3.7 contains a P2P network structured by a binary search tree. The algorithm used to organize the topology of the virtual layer offers an efficient way to identify nodes and routing requests to corresponding resources. However, compared to the unstructured network, keeping the network structured results in an increase in workload and makes structured networks harder to implement. In order to route network traffic efficiently, most algorithms require the peers to maintain a list of available resources from active neighbours. Such information needs to be continuously updated when new peers join the network or current peers leave the network. Therefore, maintaining this information will make the system less robust and more fragile when large numbers of nodes are frequently joining and leaving the network [35, 36].

3.3 Connection Technology

P2P network can be used to solve the application layer's communication issues. The actual data packs still need to be carried by the physical media. There are several technologies available to sustain physical level communication between devices, such as Wi-Fi, Wi-Fi Direct, Bluetooth, and NFC. Since NFC is especially

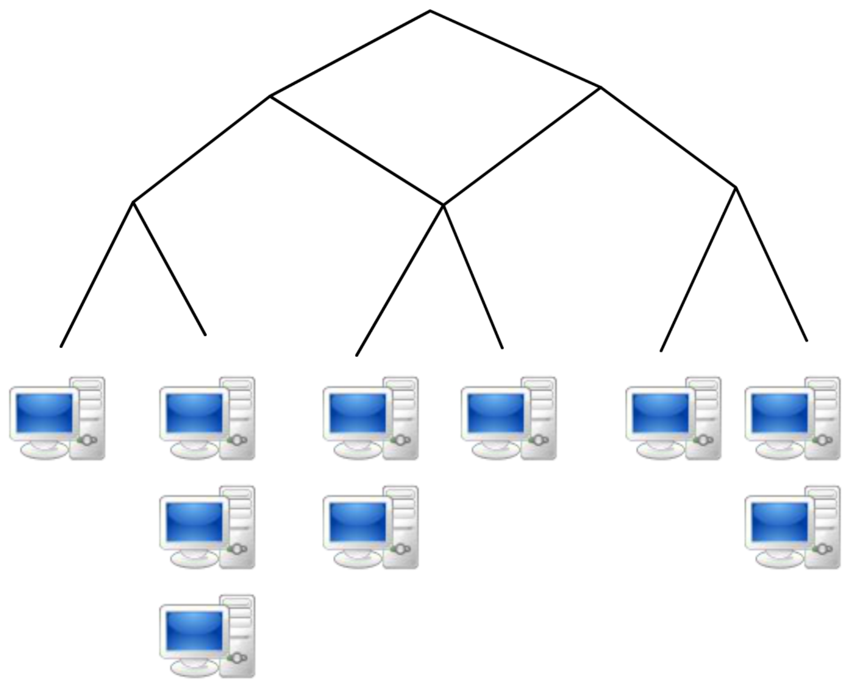


Figure 3.7: An Example of a Structured P2P Network

designed for communication between devices within up to 20cm range [37], it is definitely not suitable for the scenarios of this study. Therefore, with the exception of NFC, this study will review all the mentioned wireless technologies in the remaining sections.

3.3.1 Wi-Fi Network

Wi-Fi is the name of a popular wireless networking technology that enables digital devices to connect to a wireless local area network (WLAN) using radio waves in 2.4GHz and 5GHz frequency. The Wi-Fi Alliance uses Wi-Fi as the trademark for certifying the wireless networks based on IEEE 802.11 standards which define the detailed requirement of WLAN. Because of the strong interconnection between Wi-Fi and IEEE 802.11 standards, Wi-Fi is generally used as the synonym for WLAN and IEEE 802.11 standards. WiFi has been widely supported by most modern digital devices such as personal computers, mp3 players, smartphones, tablet computers, printers, laptops, and video-gaming consoles [38].

The Wi-Fi Alliance was founded in 1999 by several visionary companies as a non-profit association that aimed to develop a new wireless networking technology that could provide the best user experience. The Wi-Fi Alliance was initially named the Wireless Ethernet Compatibility Alliance (WECA) but changed to Wi-Fi Alliance in 2002. In 2013, the Wi-Fi Alliance was merged with the Wireless Gigabit Alliance (WiGig) to facilitate the development of 60GHz high-speed wireless network [39].

Release Date	Protocol	Frequency (GHz)	Bandwidth (MHz)	Data Transmitting Speed (Mbit/s)	Approximate Range Indoor (m)	Approximate Range Outdoor (m)
1997	802.11	2.4	22	1,2	20	100
1999	802.11a	5	20	6, 9, 12, 18, 24, 36, 48, 54	35	120
1999	802.11b	2.4	22	1, 2, 5.5, 11	35	140
2003	802.11g	2.4	20	6, 9, 12, 18, 24, 36, 48, 54	38	140
2009	802.11n	2.4/5	20	400 ns GI : 7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65, 72.2	70	250
				800 ns GI : 6.5, 13, 19.5, 26, 39, 52, 58.5, 65		
			40	400 ns GI : 15, 30, 45, 60, 90, 120, 135, 150		
				800 ns GI : 13.5, 27, 40.5, 54, 81, 108, 121.5, 135		
2012	802.11ad	60	2,160	Up to 6,756 (6.59 Gbit/s)	60	100
2013	802.11ac	5	20	400 ns GI : 7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65, 72.2, 86.7, 96.3	35	NA
				800 ns GI : 6.5, 13, 19.5, 26, 39, 52, 58.5, 65, 78, 86.7		
			40	400 ns GI : 15, 30, 45, 60, 90, 120, 135, 150, 180, 200		
				800 ns GI : 13.5, 27, 40.5, 54, 81, 108, 121.5, 135, 162, 180		
			80	400 ns GI : 32.5, 65, 97.5, 130, 195, 260, 292.5, 325, 390, 433.3		
				800 ns GI : 29.2, 58.5, 87.8, 117, 175.5, 234, 263.2, 292.5, 351, 390		
			160	400 ns GI : 65, 130, 195, 260, 390, 520, 585, 650, 780, 866.7		
				800 ns GI : 58.5, 117, 175.5, 234, 351, 468, 702, 780		
Est 2017	802.11ay	60	Unknown	Up to 20 Gbit/s	100	500

Table 3.1: Development History of Wi-Fi Network

Development History of Wi-Fi Network

IEEE 802.11 is the set of standards that describes the protocols for implementing wireless communication. The first version of IEEE 802.11 was announced in 1997, and it initially defined the fundamental standards for wireless communication in the Media Access Control (MAC) layer and the Physical layer. It was working at 2.4 GHz and provided up to 2 Mbps data transmitting speed. In 1999, its successor 802.11b was published and increased the speed to 11Mbps, which triggered the popularity of Wi-Fi. Table 3.1 shows the details of the Wi-Fi network's development history [40].

How Wi-Fi Network works

Creating a Wi-Fi network requires at least one Wireless Access Point (Wireless AP). A wireless AP broadcasts the SSID (Service Set Identifier) of the network via beacon packages to the nearby area every 100 ms. The typical stream speed of beacon packages is only 1 Mbps and the beacon package lasts for only a trifling period of time, thus the network performance is not degraded. When a client discovers the SSID and decides to join the network, it will send out a joining request to the AP. If it is a secured wireless network, the client needs to provide the access password to the AP for verification as well. Once the AP has received the joining request from the client with correct password, it will assign an IP address to the client if DHCP is configured or forward the request to the higher level router to request a valid IP. Once a valid IP is issued, the client is successfully connected to the network and will be able to communicate to any other client in the network via any network protocol [38, 41].

Pros and Cons of Using Wi-Fi in mobile environment

According to the characteristic of applications for mobile environment, the advantages and disadvantages of using Wi-Fi network for developing mobile apps are evaluated as follows:

Advantages:

- Wi-Fi technology is extensively supported across multiple platforms. Most modern digital devices, such as personal computers, laptops, and cellphones, are equipped with built-in wireless adapters.
- Wi-Fi technology has a well-supported and friendly cross-platform programming interface. Applications can communicate via traditional protocols like TCP/IP and UDP regardless as to whether the network is wired or wireless. Because there are several different major platforms (e.g., Android, iOS, Windows Phone, winCE, Linux) dominating different areas of the mobile environment, cross-platform features will significantly simplify the development.
- The data rate is sufficient enough to support fast data transmission.
- It has a variety of features to adapt different scenarios and requirements. For example, Wi-Fi Protected Access (WPA) encryption mechanism can secure the entire wireless network, and new Wireless Multimedia Extensions (WMM) protocols for quality-of-service makes Wi-Fi more suitable for latency-sensitive applications which are very common in the mobile environment [42].

Disadvantages:

- Many factors affects the effective range of a Wi-Fi network. For instance, walls or any impediments may drastically impact the network quality of a Wi-Fi network. Increasing the coverage of a Wi-Fi network requires the placement of more access points and wireless repeaters, which will cause significant difficulty in the maintenance of the network [41].
- Interference may occur if two Wi-Fi networks are using conflict channels or bandwidth. In a dense community, other radio signals with the same frequency may also cause pollution to a nearby Wi-Fi network. Accordingly, the network stability of a mobile environment cannot be guaranteed.
- As mentioned in section 3.3.1.2, at least one Wi-Fi access point is needed to set up and serve a Wi-Fi network. However, such limitation will make setting up a Wi-Fi network impossible or impractical. Recalling the IoT example illustrated in the Coordination Model section, it was impossible to place Wi-Fi routers and access points to such a vast area, like a farmer's field. Hence, using the Wi-Fi network will limit the scope of developing mobile applications.

3.3.2 Wi-Fi Direct

Wi-Fi Direct, initially named Wi-Fi P2P, is a new technology defined by the Wi-Fi Alliance aiming to enable devices to directly connect with each other without requiring the presence of external access points. Wi-Fi Direct is still compatible with previous Wi-Fi standards, meaning that internet browsing, file transfer, and data sharing are still possible. Devices under the Wi-Fi Direct network can communicate with one or more devices simultaneously at the typical speed of a Wi-Fi network. In a Wi-Fi Direct network, only one device needs to be compliant with the Wi-Fi Direct standard to initiate the peer-to-peer network. All other devices can connect together via the same procedure as connecting to a conventional Wi-Fi network [43].

Background

According to the Wi-Fi Direct white paper published in 2014, more than 2 billion Wi-Fi supported devices have been manufactured and sold to the market in 2013, which is 30% more than in 2012. These numbers are expected to rapidly rise over the next 5 years. The Wi-Fi Alliance has issued certifications to over 20,000 products from over 650 member companies. Therefore, the Wi-Fi Alliance believes that Wi-Fi technology has already stepped into its second phase [44].

However, conventional Wi-Fi technology requires at least one controller device, known as the wireless access point, to set up the network. These wireless access points will be responsible for three fundamental tasks: [45, 44]

- Provide physical support for wireless networking, including broadcasting and validating new joining requests;
- Bridging and routing the communication between devices;
- Properly managing the network topology when devices connect and disconnect;

As shown on the left side of figure 3.8, the access point acts like a central hub that transfers all the network traffic to desired destinations in the conventional Wi-Fi network. Although a client can still virtually connect to each other, it cannot physically establish a direct connection to other clients because all messages need to go through the access point. To overcome such limitations, the Wi-Fi Alliance announced the Wi-Fi Direct standards in 2010 that enable devices to create direct connections between Wi-Fi client devices without having the presence of conventional Wi-Fi infrastructure [46], as shown on the right side of figure 3.8. The introduction of Wi-Fi Direct contributes a vital foundation to greatly improve the portability and flexibility of applications.

While still retaining the strengths of traditional Wi-Fi such as performance, security, and ease of use – Wi-Fi Direct enhances the user experience by providing natural physical level support for Peer-to-Peer networking. Rather than connecting to an infrastructure network first, and then virtually connecting to another device, devices can directly request the services exposed by other nodes. This is useful in many

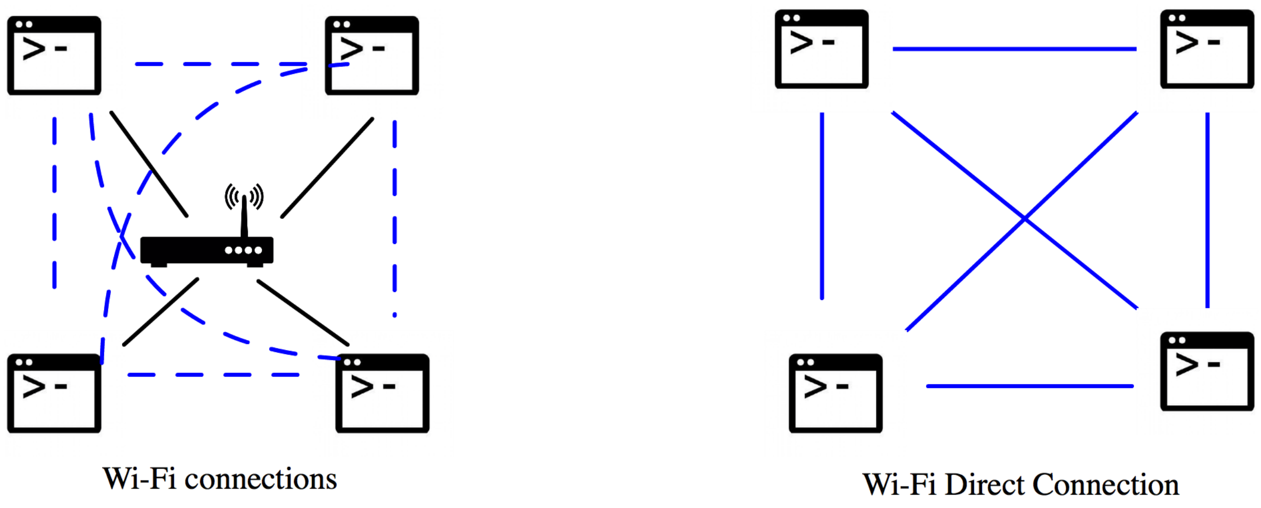


Figure 3.8: Connections Comparison between Conventional Wi-Fi and Wi-Fi Direct

scenarios. For example, a hiking tourist can instantly print a photo he just took by sending it directly to the mini-printer in his pocket without requiring a stable Wi-Fi access, which is impossible in a mountainous area.

By 2014, more than 1.1 billion Wi-Fi Direct-enabled devices were shipped worldwide. As shown in figure 3.9, it is predicted that Wi-Fi Direct will be embedded in 80 percent of all Wi-Fi devices by 2019. Moreover, since Wi-Fi Direct is compatible with the conventional Wi-Fi, which means Wi-Fi Direct-certified devices support connections with existing legacy Wi-Fi devices, it brings the capability for performing software level P2P to billions of existing devices [47].

Features

Wi-Fi Direct naturally supports P2P network and has the following benefits: [46, 44]

- Mobility & Portability

  The Wi-Fi Direct technology enables devices to connect to each other without requiring an external Wi-Fi router or access point. A Wi-Fi Direct network can be created at any time and any where.

- Instant Utility

  As long as a user has a Wi-Fi Direct-certified device in hand, all of his other Wi-Fi enabled devices will be able to join the Wi-Fi Direct network.

- Ease of Use and Efficient Network

  Wi-Fi Direct provides a productive mechanism to identify the nodes and services of a connection before physically connecting. For example, when a user wants to print a document, he can know which node in the current Wi-Fi Direct network has printing capability without establishing a connection to every other node in the network.

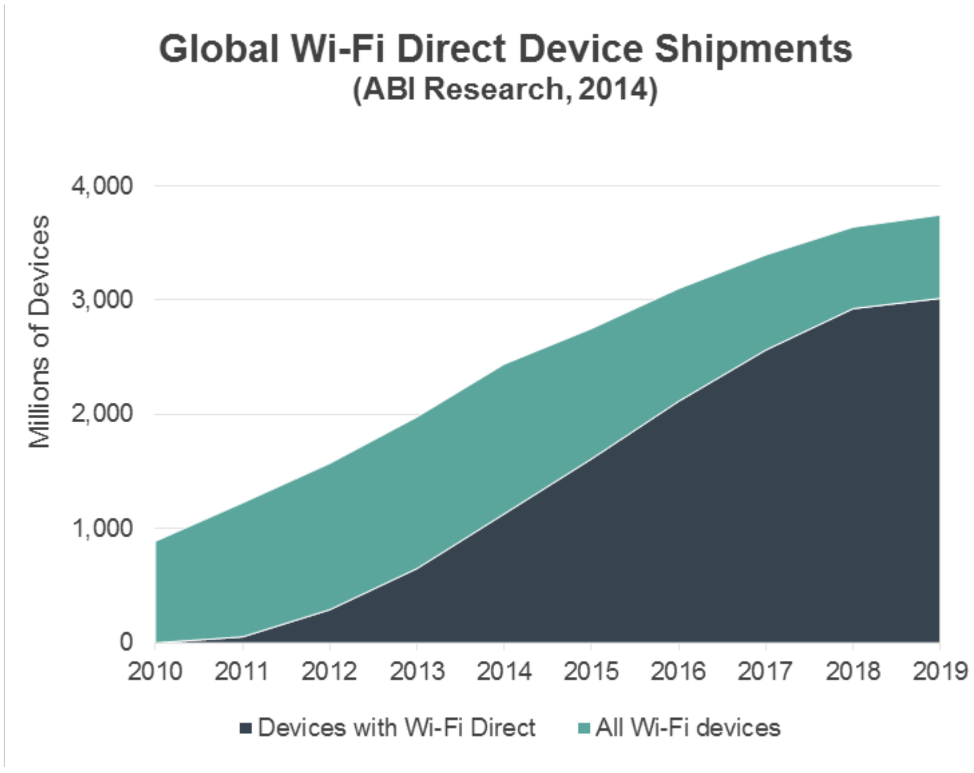


Figure 3.9: Global Wi-Fi Direct Device Shipment

- Simple Secure Connection

  Besides the security mechanism in traditional Wi-Fi technology like WPA/WPA2, Wi-Fi Direct devices can also use Wi-Fi Protected Setup (WPS) technology to make creating a secure connection between devices simple and efficient. A user can launch the authentication procedure by either pressing a button on both sides or using other touch-less technology like NFC for automated authentication.

- Flexibility

  Wi-Fi Direct technology presents numerous possibilities for developing utilities that addresses users' needs. For instance, with the help of Wi-Fi Direct, users can share files, print photos, play media, and enjoy innovative n-screens interaction among multiple devices.

- Programmability

  Wi-Fi Direct defines several cross-platform and easy-to-use APIs for programmers to develop applications utilizing the features of Wi-Fi Direct.

Technology Basics

According to the specification, a Wi-Fi Direct device is capable of either a P2P connection or a Wi-Fi connection, and must support the Wi-Fi Protected Setup [44]. Wi-Fi Direct devices are connected by forming *Groups*. The topology of a *Group* can be one-to-one or one-to-many. One of the members in a group needs to be elected as the *Group Owner* (GO) once a Wi-Fi Direct network has been created. The *Group Owner* is responsible for controlling and managing the network. The *Group Owner* decides which device is allowed to join the network and grants proper permissions to members according to given rules. The *Group Owner* also functions as the AP to legacy clients of the conventional Wi-Fi network, meaning that it should be compatible with Wi-Fi devices that are not compliant with the Wi-Fi Direct standard. Legacy clients can only function as clients within a group and can never be the Group Owner [43].

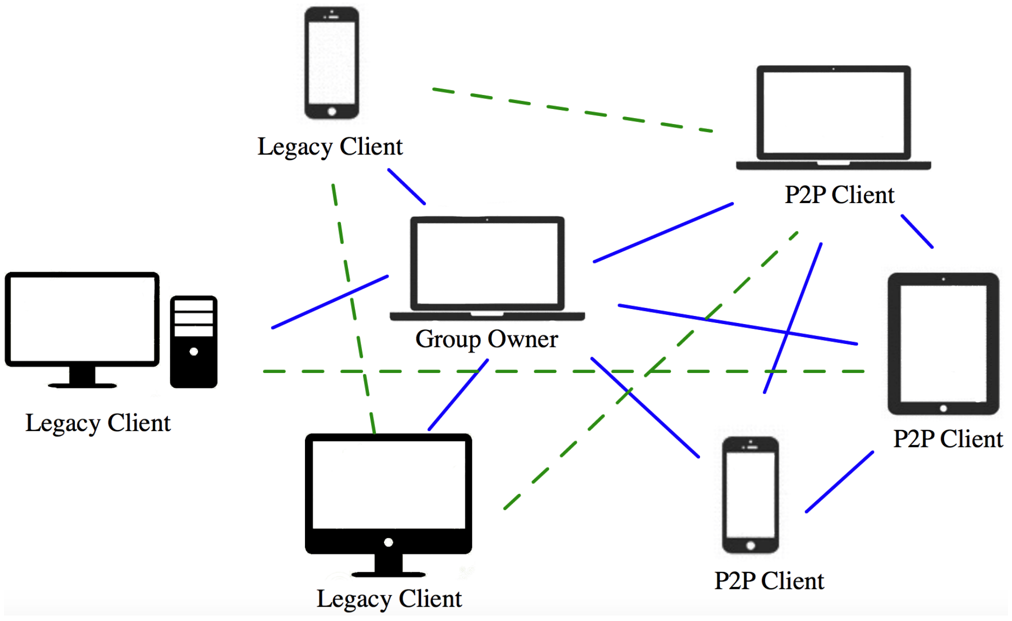


Figure 3.10: An Example Wi-Fi Direct Network

Figure 3.10 gives an example of a typical Wi-Fi Direct network. Four P2P clients are directly connected to each other and one of them has been elected as the *Group Owner* to coordinate the network. All other legacy clients of conventional Wi-Fi can establish connections to the *Group Owner* and then create virtual connections to other nodes including the P2P clients on top of the physical connections.

Item	Introduction	Mandatory	Optional
Device Discovery	Mechanism for detecting nearby Wi-Fi Direct devices and exchange device information	X	
Service Discovery	Mechanism for discovering the services provided by nearby Wi-Fi Direct devices. Should be able to exercise prior to establishing the real connection.		X
Group Owner Election	Capability to negotiate with other participating Wi-Fi Direct devices to determine a *Group Owner*	X	
Invitation	To allow a Wi-Fi Direct device to invite other Wi-Fi Direct devices to join an existing group		X
Client Discovery	Enabling a Wi-Fi Direct device to detect other devices (regardless it is a Wi-Fi only or Wi-Fi Direct devices) in an existing group.	X	
Power Management: P2P Power Save and P2P-WMM Power Save	Same as the Power Save and WMM-Power Save mechanisms in Wi-Fi.	X	
Power Management: Notice of Absence	Technique that enables the *Group Owner* to reduce the power consumption by notifying other participating devices a planned absence	X	
Power Management: Opportunistic Power Save	Similar to Notice of Absence mechanism, it is a technique that enables the *Group Owner* to reduce the power consumption by entering a doze state while the network is idle.	X	
Persistent Groups	Mechanism that allows a previously established group to be re-invoked at a future time without re-provisioning		X
Concurrent Connections	Capability of a Wi-Fi Direct device to simultaneously maintain multiple connections from either Wi-Fi Direct devices or legacy clients.		X
Multiple Groups	Mechanism that allows a Wi-Fi Direct to join multiple groups at the same time.		X
Cross-Connection	Allows Wi-Fi Direct devices provide traditional Wi-Fi access to other devices in the group	X	

Table 3.2: Detail Specification of Wi-Fi Direct

All Wi-Fi Direct certified devices must be capable to control a group, i.e., being the *Group Owner*, and must be able to participate the negotiation with all involved devices to elect the *Group Owner* when forming a group. Wi-Fi Direct specification also requires devices to support client discovery and power management mechanisms. Other features such as P2P service discovery and concurrent infrastructure connections are optional when designing Wi-Fi Direct certified devices. Table 3.2 contains the detail specification for different features of Wi-Fi Direct. Complementary explanation for the items in the table are as follows:

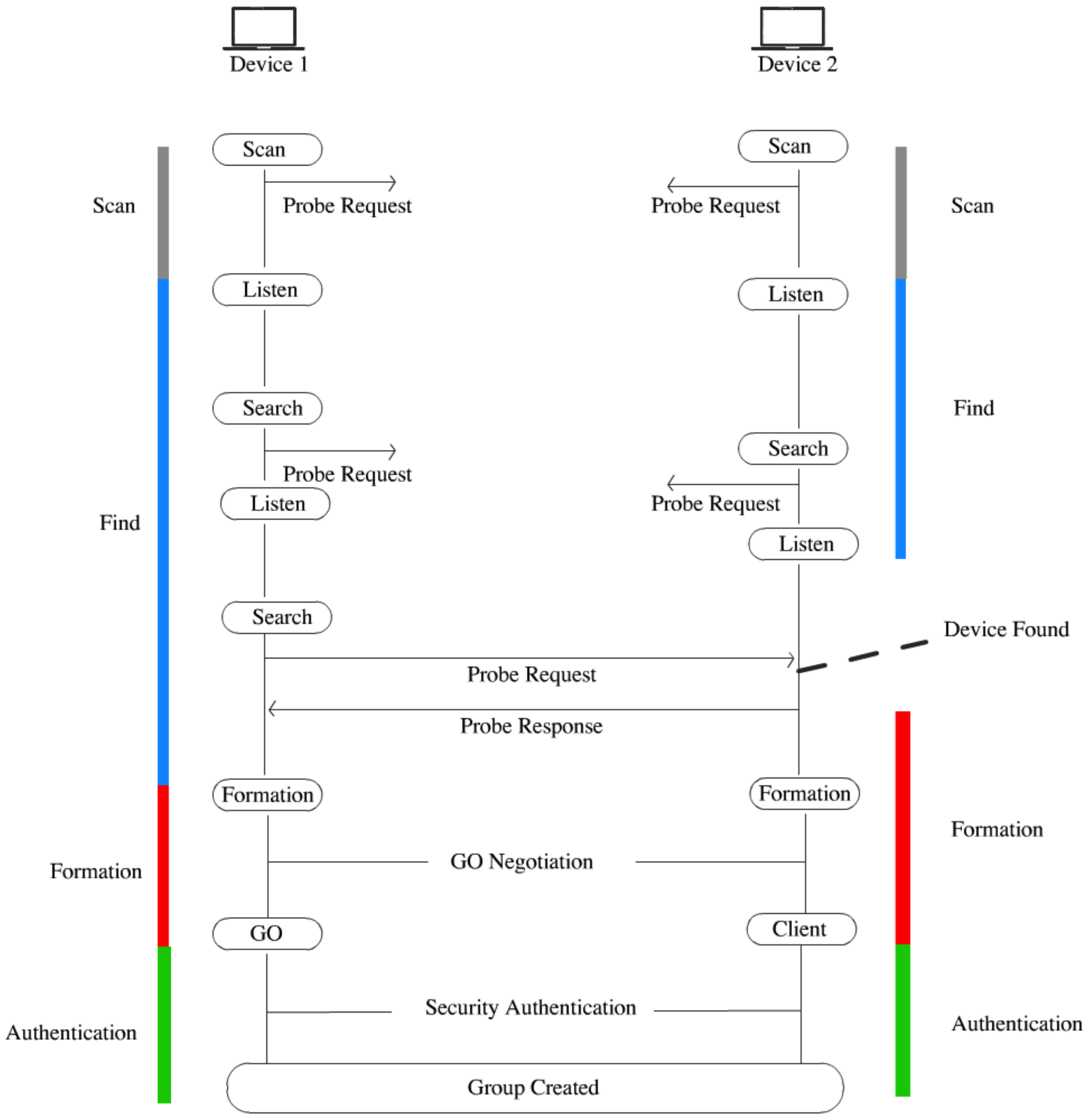


Figure 3.11: Detailed Procedure of Forming a Wi-Fi Direct Network

- Device Discovery

  Device discovery is a mechanism for detecting other Wi-Fi Direct devices in order to establish connections. Device discovery is comprised of 4 steps: Scan, Find, Formation, and Authentication. A device will first try to scan for an existing group and, if found, it will attempt to join the existing group. Otherwise, the device will enter the *Find* phase to seek for other Wi-Fi Direct devices. If a Wi-Fi

Direct device is found, the device will then retrieve the information of the *Group Owner* or negotiate to elect a *Group Owner*. Otherwise, the device will create a new group and make itself the *Group Owner*. Either legacy security mechanism like WPA/WPA2 from the traditional Wi-Fi or WPS can be utilized to secure the newly formed Wi-Fi Direct network.

Figure 3.11 illustrates the detailed procedure of device discovery. As shown in figure 3.11, Device 1 (D1) and Device 2 (D2) become online at the same time. At first, they both start scanning for existing Wi-Fi Direct devices by sending out probe request. Because none of them are listening (only a listening device can receive probe requests and send back a reply), they won't be able to discover each other, which means a new group needs to be created. The *scan* phase will last for a certain period of time, then they will both enter the *find* phase. In the *find* phase, both D1 and D2 will be continuously switching between *listen* and *search*, and each state will last for a random period of time. Therefore, eventually one of them will be in the search state while the other is listening, as shown in the middle of figure 3.11 where D1 was on its second run trying to search for a signal while D2 remained in the listening state. Once D2 receives the probe request sent by D1, it will return a probe response to D1. This indicates that D2 is found and then they can both proceed to the *formation* phase. They will launch the GO negotiation procedure to decide which of them to be the *Group Owner*. If D1 is elected to be the *Group Owner* and D2 is to become the client, then D1, the owner of the group, will start the security authentication procedure to verify whether D2 should be permitted to join. Finally, once D2 is granted the permission to access the network, a new group is successfully created.

- Service Discovery

  Service Discovery is an optional P2P feature targeting to support the advertisement of services provided by the participating devices within a Wi-Fi Direct network. Popular examples of such services include Bonjour, UPnP, and Web Service Discovery. The procedure of service discovery can be launched at anytime (even before a real connection is created) by a Wi-Fi Direct device. Although it is an optional feature, it has been widely supported by almost all Wi-Fi Direct certified devices.

- Power Management

  Efficient use of power is extremely important for Wi-Fi Direct devices as they are battery-operated in many cases. The Wi-Fi Direct specification contains several power management mechanisms for reducing the power consumption, while avoiding any impacts to the performance of the network.

  Besides the *P2P Power Save* and *P2P-WMM-Power Save* mechanisms adapted from the Wi-Fi standard, Wi-Fi Direct introduces two new power saving strategies: *Notice of Absence* and *Opportunistic Power Save*.

  The *Notice of Absence* mechanism makes it possible for a *Group Owner* to announce a planned absence. Such absences can be either single or periodic, and when other participants in the group have received the announcement, they can then prepare themselves so that the network will keep functioning when the *Group Owner* is "having a rest".

Opportunistic Power Save means that when the *Group Owner* detects that the entire group is in an idle state with no activity among devices, it will enter the doze state so that the power it uses to maintain the network signals can be saved.

However, no matter which power saving mechanism is used, Wi-Fi Direct requires the *Group Owner* to be periodically available so that the device discoverability can be guaranteed, and devices that are attempting to connect to the network need to be aware that power management mechanisms are in use. Both *Notice of Absence* and *Opportunistic Power Save* places the requirement that all of the devices in the network should be Wi-Fi Direct enabled devices. If legacy devices are present, only power management mechanisms designed for traditional Wi-Fi architecture can be employed.

Example Applications

Wi-Fi Direct has been widely used in people's daily life. Here are some typical examples of Wi-Fi Direct.

- AppleTV [48]

  AppleTV utilizes Wi-Fi Direct to implement the AirPlay feature. AirPlay allows users to wirelessly access a big screen, like an HDTV, to display the content of their iPhone, iPod, iPad, and Macs.

- AirPrint [49]

  AirPrint is quickly becoming one of the most dominant ways for users to connect to their printers. A user is able to directly connect his cell phone or computer to his printer without having them on the same wireless network. AirPrint missions are transferred via *Bonjour service* which is essentially based on Wi-Fi Direct technology.

- Jott Messenger [50]

  Jott Messenger is an app that gives users a way to make friends with nearby people and stay connected with them. It utilizes Wi-Fi Direct to enable a user to discover nearby users and send out messages. People from similar locations, such as a university, can easily form a group. Wi-Fi Direct helps the users to secure their privacy because none of their chatting records will be exposed to the internet.

- Spaceteam Card Game [51]

  Spaceteam is a popular cooperative party game. It wirelessly connects the players together without requiring them to participate in the same WLAN or to have internet access. Players use their cell phones or tablets to play and the real-time result will be distributed to every participant instantly. Such features of Spaceteam are also driven by the Wi-Fi Direct technology.

3.3.3 Bluetooth

Bluetooth is a short-range wireless data transmitting technology operating in the 2.4GHz frequency band [52]. Bluetooth was first invented as a wireless alternative to RS-232 data cables by the telecom company Ericsson in 1994, and it is now managed by the *Bluetooth Special Interest Group (Bluetooth SIG)* [53]. The latest stable version of Bluetooth is in its 4th generation known as the *Bluetooth Low Energy (BLE)* [54]. Bluetooth 5 was recently announced in June 2016 and is expected to be adopted into production in early 2017 [55].

History of Bluetooth

In 1998, five companies including Ericsson, IBM, Intel, Nokia, and Toshiba formed the Bluetooth SIG to develop the first global version of the Bluetooth standard, which aims to promote a solution for short-range wireless communication operating in the unlicensed 2.4GHz ISM(Industrial, Scientific, Medical) frequency band. The IEEE standardized Bluetooth as IEEE 802.15.1 in 2002, but this standard is no longer maintained. The Bluetooth SIG is the only official organization that manages the Bluetooth specification, qualification program, and trademarks.

To facilitate the spread of the newly developed Bluetooth technology, the Bluetooth SIG decided to make all of the explicitly included features in the Bluetooth specification royalty-free to all the adopter members who utilize Bluetooth technology in their new products. The Bluetooth SIG has three levels of corporate membership including Promoter members, Associate members, and Adopter members. The Bluetooth community has been consistently growing since it was established.The Bluetooth SIG was started with 70 members and later on this number grew to 3,000 by the year of 2001, and as of this writing, there are approximately 30,000 members in total [56].

Year	Version	Transmission Rate	Approximate Range(m)
1999	Bluetooth 1.x (1.0, 1.1)	723.2kbit/s	10
2003	Bluetooth 1.2	723.2,kbit/s	10
2004	Bluetooth 2.0 + EDR	2.1 Mbit/s	10
2007	Bluetooth 2.1 + EDR	3 Mbit/s	10
2009	Bluetooth 3.0 + HS	24 Mbit/s	10
2010+	Bluetooth 4.x(4.0, 4.1, 4.2)	24 Mbit/s	60
2016	Bluetooth 5	50Mbit/s	240

Table 3.3: Bluetooth History Versions

Table 3.3 gives a brief introduction about the development history of Bluetooth technology. Until now, there have been five generations and ten main iterations of Bluetooth technology, with all iterations before Bluetooth 4.0 can be classified as the *Classic Bluetooth (Classic BT)* and iterations since Bluetooth 4.0 are called *Bluetooth Low Energy (BLE).* The main improvements of each iteration will be listed as follows: [57, 52, 58, 59, 60, 61]

- Bluetooth v1.0

  This is the initial version of Bluetooth. However, it was not a production ready standard because it still had many problems for manufacturers to implement. For example, it placed an uncommon mandatory requirement that Bluetooth hardware device address needs to be known even in the connecting process, which prevented many services from using Bluetooth.

- Bluetooth v1.1

  The Bluetooth v1.1 is first successful member of the Bluetooth family. It fixed many errors and problems found in the previous version and added support to non-encrypted channels. It was ratified as the IEEE standard 802.15.1-2002 standard.

- Bluetooth v1.2

  Bluetooth v1.2 contains many important enhancements and is ratified as IEEE standard 802.14.1-2005.

  - Bluetooth v1.2 significantly reduced the time spent on the discovery and connection of nodes.
  - It also introduced the *Adaptive Frequency-Hopping spread spectrum (AFH)* to improve resistance to radio frequency interference by avoiding to occupy the crowded frequencies in the hopping sequence.
  - It introduced the idea of *Extended Synchronous Connections (eSCO)*, which allows retransmissions of corrupted packets.

- Bluetooth v2.0 + EDR (Enhanced Data Rate)

  The major improvement of this version is the introduction of the idea *Enhanced Data Rate* which has significantly increased the data transmission speed. Although the data rate can be 3 Mbit/s in theory, the practical rate is about 2.1 Mbit/s. However, it is still faster than its precursor.

- Bluetooth v2.1 + EDR

  This version improves the pairing experience and enhances the security by adding a feature called *Secure Simple Pairing (SSP).*

- Bluetooth v3.0 + HS (High Speed)

  As indicated in the name, one of the most important features of this version is the great improvement of the data transfer speed. When a Bluetooth v3.0 is working in HS mode, the data transfer speed can be up to 24 Mbit/s theoretically. This is accomplished by involving an 802.11 WLAN for high-speed data transmission while the traditional Bluetooth link was only used for negotiation and establishment. This new feature was called the *Alternative MAC/PHY (AMP)*.

Also, Bluetooth v3.0 enhanced the power control by removing the open loop power control and clarifying ambiguities of the modulation schemes added for EDR.

- Bluetooth v4.0 - Bluetooth Low Energy

  The release of Bluetooth v4.0 introduced the new concept referred to as *Bluetooth Low Energy (BLE)*, which brought the development of Bluetooth technology to a brand new era. The Bluetooth v4.0 consisted of 3 parts: 1. the Classic Bluetooth which supports the legacy Bluetooth protocols; 2. the Bluetooth High Speed which is kept the same as v3.0; 3. the BLE.

  BLE was originally introduced under the name *Wibree* by Nokia in 2006, and was merged into the Bluetooth standard in 2010. BLE came with an entirely new protocol stack to support very low power consumption and rapid establishment of simple links. In late 2011, the Bluetooth SIG announced *Bluetooth Smart Ready* for hosts and *Bluetooth Smart* for sensors to be the public face of BLE.

  Chips designed for BLE can be implemented under two modes: the single mode and the dual mode, where the single mode only supports the low energy protocol stack and the dual mode requires implementation for both low energy protocol and classic Bluetooth protocols including the HS feature.

- Bluetooth v4.1

  The main features added by v4.1 were:

  – Allowed devices to act as multiple roles simultaneously;

  – Allowed co-existence of other mobile wireless services;

  – Limited the discovery time;

  – Low duty cycle directed advertising and fast data advertising interval;

  – Introduced BLE link layer topology;

- Bluetooth v4.2

  Bluetooth v4.2 introduced some features specially designed for IoT. For example, it added an IPv6 connection option for *Bluetooth Smart* so that IoT implementations can be supported.

- Bluetooth v5

  Bluetooth v5 was recently announced in June, 2016. Bluetooth v5 mainly focuses on supporting IoT technology and it guarantees to *quadruple the communication range and double the data transmission speed*.

Technique Basics

We will go through the technique basics of the Bluetooth technology in two sections: the ***Classic Bluetooth*** and the ***BLE***.

Classic Bluetooth

1. Topology

   Bluetooth uses master-slave structure under scatternet topology. A scatternet is an ad-hoc network consisting of two or more piconets, which are the collections of Bluetooth-enabled devices communicating with each other. A piconet requires one device to be the master with all other members acting as slaves to it. The master will be responsible for providing certain services, while the slaves will consume the services. The master of a piconet will assign a unique *active member address (AM_ADDR)* to each slave so that it can identify the slaves. An AM_ADDR is expressed by 3 binary bits and "000" is reserved for broadcast, meaning that a master of classic Bluetooth can only have up to 7 slaves.

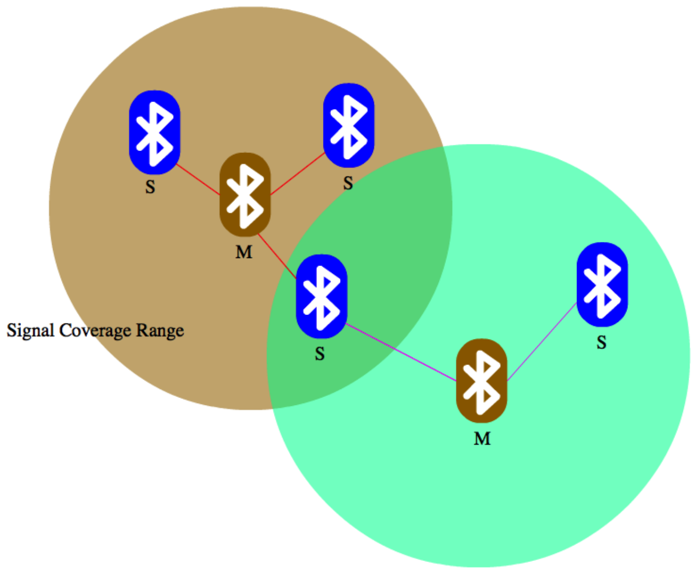


Figure 3.12: Bluetooth Scatternet

   Figure 3.12 gives an example of the Bluetooth scatternet. Two piconets can form a scatternet when a member within both ranges participates as a slave in the other piconet. Therefore, the scatternet mechanism enables Bluetooth to support communication for more than eight devices.

2. Protocol Stack

   Figure 3.13 depicts the classic Bluetooth protocol stack. The *Link Management Protocol (LMP)*, the *Logical Link Control and Adaptation Protocol (L2CAP)* and the *Service Discovery Protocol (SDP)* are mandatory in the classic Bluetooth, .

   The LMP is responsible for setting up and managing the physical radio link between devices. The

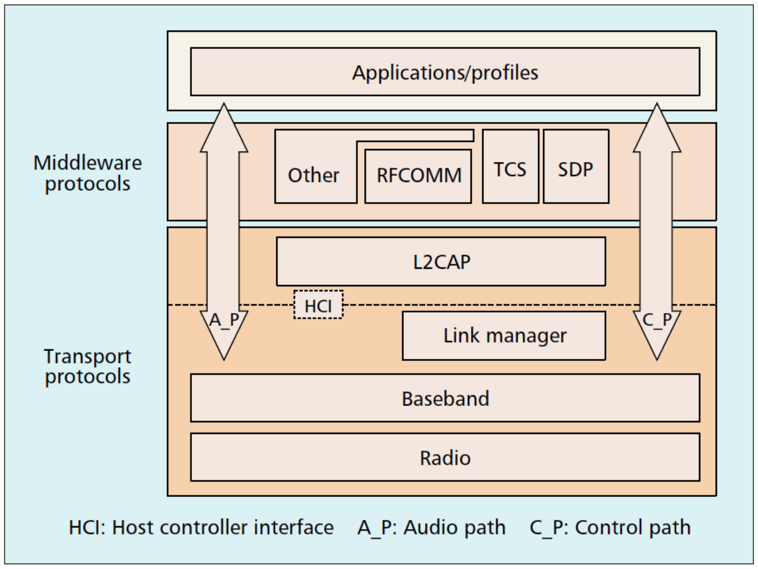


Figure 3.13: Classic Bluetooth Protocol Stack

L2CAP is designed to provide support for higher-level protocol multiplexing, packet segmentation/re-assembly, and passing the quality of service information. The SDP performs the service discovery function of Bluetooth. RFCOMM, short for *Radio Frequency Communications*, and it is the cable replacement protocol that is designed to replace the RS-232 cables. The *Telephony Control Protocol (TCS)* functions the voice and call control between Bluetooth devices.

3. Connection Establishment

   A Bluetooth device needs to be set to the discoverable mode to accept new connections. A discoverable Bluetooth device will broadcast information including the device name, class, list of services, and technical report to the nearby area for a certain amount of time. If another Bluetooth device discovered the discoverable device and decided to connect, it will send out a connection request to it. A *Pairing* procedure will be launched if this is the first time the devices have connected to each other. Otherwise, these two devices will be connected together to form a piconet and the discoverable device will be the master.

4. Pairing and Bonding

   Like the WPA/WPA2 mechanism in WLAN, pairing is the mechanism invented to secure Bluetooth connections and prevent unauthorized access. A *pairing* can either be triggered manually by the user or automatically when attempting to request a service. A *bonding* is generated once a *pairing* is finished, which means that the two devices are successfully paired. Two devices pair with each other by creating a shared secret called a *link key*. Both devices are paired immediately if they have previously stored the *link key*. Otherwise, a new link key will be generated and then cryptographically shared to the other devices to create a new pairing.

BLE

BLE uses the same spectrum as classic Bluetooth but a different set of channels. Moreover, *BLE* has a different protocol stack that results in a faster data transfer speed and lower power consumption.

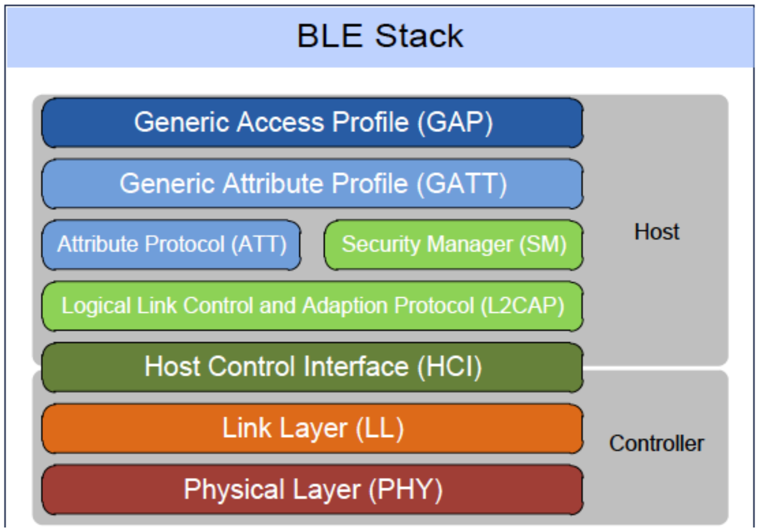

BLE Stack
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Attribute Protocol (ATT)
Security Manager (SM)
Logical Link Control and Adaption Protocol (L2CAP)
Host Control Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)
Host
Controller

Figure 3.14: BLE Protocol Stack

Figure 3.14 depicts the BLE protocol stack. The *GAP* is responsible for device discovery and link management. It also defines four basic roles in the application layer of BLE: Broadcaster, Observer, Peripheral, and Central.

The *Broadcaster* will continuously broadcast advertising events to the nearby area. Broadcasting provides a way for the BLE device to transmit data to more than one device at a time. The *Observer* receives the advertising events from the broadcaster. A device operating in the *Peripheral* role will periodically send connectable advertising packets and accept new connections. A peripheral device will function as a slave in the *Link Layer*. A device in the *Central* role will be the master node and initiates the connections to peripherals.

The GATT defines a generic service framework using the ATT protocol layer which builds a Client/Server architecture on top of the BLE L2CAP layer. This framework defines the standard of services, characteristics, and procedures for reading, writing, notifying and indicating data. There are two roles defined in GATT: GATT Server and GATT Client, where a GATT Server will serve requests for services and characteristics from the GATT Clients.

The SMP of BLE is similar to the SDP of classic Bluetooth. The SMP is responsible for managing the pairing and bonding, data encryption and authentication, as well as choosing the pairing method based on the IO capability of the GAP central and peripheral.

3.4 REST and CoAP

3.4.1 REST

REST, short for *Representational State Transfer*, is an abstract software architecture illustrating several principles for building scalable distributed systems [62]. Its principles have been adopted to develop the excellent scalability of HTTP protocol which is the fundamental component of the World Wide Web [63]. In order to achieve higher performance and make the system more maintainable, REST involves a coordinated set of constraints applied to the design of components in the distributed system. Many of the constraints are extremely useful for coordinating distributed mobile applications, which are listed as follows: [64]

- **Identification of Resources**

  Each individual resource is represented by a unique *Uniform Resource Identifier (URI)* and is conceptually separate from the representation that will be sent to the client. Recalling the concept of ITS in the LIME coordination model, REST can be easily used to implement the ITS architecture by assigning each ITS a URI.

- **Manipulation of Resources**

  REST requires that each representation contains sufficient information to modify or delete the resource. Such constraints are often implemented by adding operations, such as GET, POST, PUT, and DELETE, and extra necessary arguments in the resource descriptions. This constraint provides inborn support for the operations in LIME, i.e., *in*, *out*, and *rd*.

- **Self-descriptive Messages**

  A self-descriptive message contains enough information to describe how to process it. The *eval* operation benefits from this constraint because it needs to know how to evaluate and execute a tuple.

- **Stateless**

  Stateless means there is no context shared between requests. Every request is independent and contains all the information required to handle it. Such a feature is ideal for the mobile environment in that connections in a mobile environment are not as stable as in a local network, and maintaining the context would be impractical in most cases.

REST seems to be ideal for mobile environment. The most recognized implementation of REST is the HTTP protocol. However, HTTP is not suitable for mobile environment for the following reasons:

1. HTTP is based on TCP protocol that uses a point to point communication model and Client/Server architecture. As explained in the previous sections, such a mechanism is not suitable for the mobile environment.

2. Most of the mobile devices are constrained devices which tend to be deeply embedded and have limited memory and power supply than traditional devices, thus, HTTP is too complex for them to maintain high performance.

Therefore, a new REST-based protocol, called CoAP, was specifically created and designed for constrained devices.

3.4.2 CoAP

CoAP, short for *Constrained Application Protocol*, is a software communication protocol specially designed for constrained devices and networks. CoAP aims to be a generic web protocol for the constrained environment, especially considering energy, computing capability, and other limitations in machine-to-machine (M2M) applications. CoAP implements a subset of REST that is common with HTTP but optimized for M2M applications, and provides natural built-in support for services discovery, multicast, and asynchronous message exchanges [65].

CoAP has been ratified as the RFC 7252 standard and its standardization is primarily done by the CoRE (Constrained RESTful Environments) group of the IETF (Internet Engineering Task Force) [66].

The procedure for handling a request in CoAP is similar to the Client/Server model in HTTP. However, the most important difference is that a node in CoAP will typically be both client and server at the same time, which means it will not only expose services but consume services provided by other nodes as well. Similar to RESTful HTTP, a request in CoAP is sent by a client to acquire a resource identified by a URI on a server.

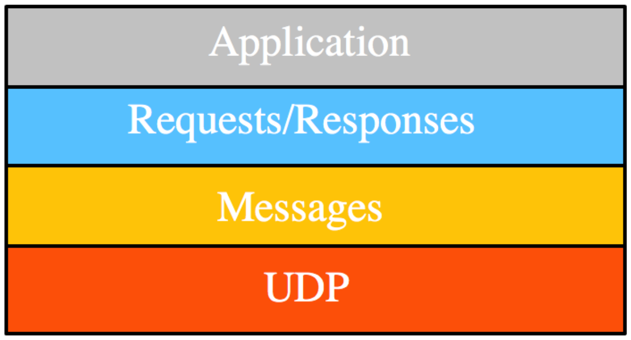


Figure 3.15: CoAP Abstract Layering

Figure 3.15 describes the layering of CoAP. All of the CoAP requests and responses from the application will be logically managed by the Requests/Responses layer using *Method* labels and *Response Code* labels in the header. All the CoAP requests and responses will finally be packed as CoAP *Messages* and transferred via UDP which implies all the interchanges will be handled asynchronously. There are four different types of messages in CoAP: *Confirmable*, *Non-confirmable*, *Acknowledgement*, and *Reset*. Requests can either be confirmable or non-confirmable, and responses can be carried in acknowledgement messages. A detailed explanation will be given later on.

Main Features

The main features of CoAP are as follows:

- It is a software web protocol that supports M2M application in a constrained environment;
- It supports reliable unicast or multicast requests based on UDP;
- It allows asynchronous message exchanges so that devices who initiated the request will not be blocked;
- It defines a very simple and small size head which can be easily and efficiently parsed;
- It is fully RESTful, which means it provides URI and Content-type support;
- It supports service auto-discovery;

CoAP Messages

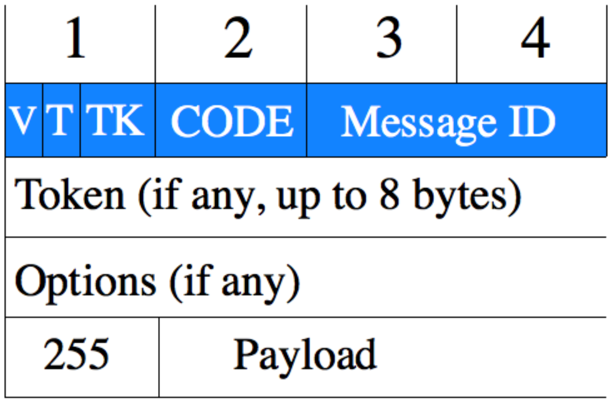


Figure 3.16: CoAP Message Format

To guarantee the performance, CoAP uses a short fixed-length binary header that costs only 4 bytes. Figure 3.16 depicts how the CoAP messages are organized. The first 2 bits in the header is an unsigned integer indicating the CoAP version. Currently, there is only one version so the value will be forced to 01. Following the 2-bits version identifier is the specification of the message type, which is also 2-bits long. Binary value 00 means *Confirmable* type, 01 means *Non-confirmable*, 10 means *Acknowledgement*, and 11 means *Reset*. A token (up to 8 bytes) can be attached after the header to correlate requests and responses. If a message contains a token, then the 4 bits after *type* field will be an unsigned integer specifying the length of the token. Otherwise, these bits will be set to zero. The second byte of the header indicates the message code, and it is split into two parts. The first part occupies 3 bits to indicate whether this message is a request (0), a success response (2), a client error message (4), or a server error response (5). Other values are reserved for future updates. The other 5 bits form the second part. These 5 bits are used to indicate the request method (GET, POST, PUT, or DELETE) in case of a request, and the response code when the message is a response. The last 2 bytes of the header is a 16-bit unsigned integer in network byte order indicating the Message ID. Message IDs are used to identify messages, detect duplicate messages, and match responses with their corresponding requests. CoAP also allows the user to attach some options to a message after the token field. Similar to the options in HTTP, the option field in CoAP is used to point out some extra attributes of the message such as Max-Age, Content-format, and URI. CoAP uses 8 consecutive bits (set to 1) to indicate the start of the payload.

If a request is set to *Confirmable*, then the request will be periodically resent until an *Acknowledgement* message with the same *Message ID* is received from the designated endpoint. A *Non-confirmable* request means it does not require a reliable transmission. The recipient may reply to the sender with a *Reset* message when the request can not be processed.

3.5 Distributed Hash Table and Consistent Hashing

3.5.1 Distributed Hash Table

A *Distributed Hash Table (DHT)* is a decentralized distributed system that supports accessing the data in a way similar to hash table. Same as the traditional hash table, data is organized as key-value pairs in DHT. The most distinguished feature of DHT is that all the key-value pairs are stored among multiple nodes and changing the set of the participating nodes will not significantly impact the system.

The foundation of DHT is an abstract key-space which will be split to all the participants according to specific key-space partitioning scheme. An overlay network is designed to describe the network topology and manage the key-space. Different implementations of DHT can have the same network topology, but they must have varying key-space partitioning schemes.

The typical procedure of storing and retrieving data in DHT is depicted in figure 3.17. The first step of saving and finding a key-value pair (k,v) is to generate the hash value of k using a designated hashing function. After that, a mapping function will try to ask a set of nodes according to the key-space partitioning scheme to find out the node which is responsible for hosting k. Saving and Finding request will be then directly sent to the node [67].

DHT provides a way to make a distributed system become scalable, autonomic, and resilient. The most popular designs of DHT is *consistent hashing*.

3.5.2 Consistent Hashing

Consistent Hashing was first introduced by MIT when developing a distributed caching solution that supports dynamic joins and leaves of servers. A common way for the key-space partitioning mechanism to calculate the destination of a given key is to calculate the result of *key%n (key mod n)*. However, such mechanism

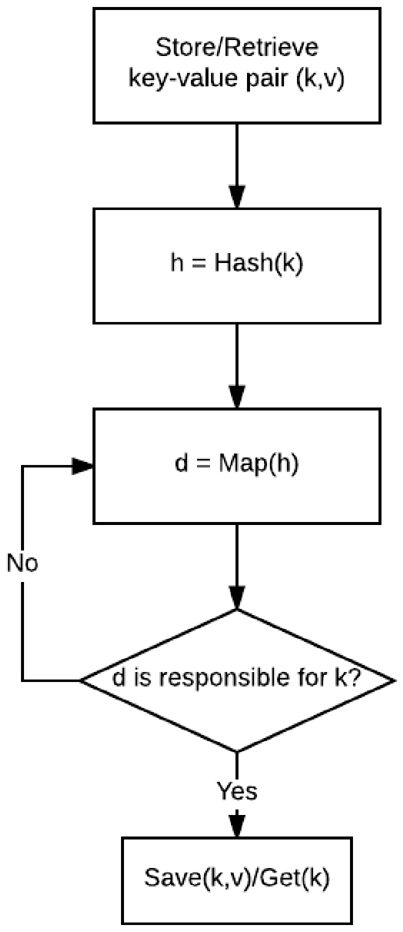


Figure 3.17: Procedure for Saving and Finding data in DHT

might cause severe performance problems because all keys need to be remapped when n changes. Consistent Hashing provides a way to guarantee that if the size of the hash table changes, then only k/n keys need to be re-shuffled, where k is the number of keys and n is the size of the hash table. Therefore, Consistent Hashing is remarkably suitable for the mobile environment where nodes are frequently connecting and disconnecting [68].

Consistent Hashing maps each key to a point on an abstract circle called the *consistent hashing ring*, and each available nodes will be randomly mapped to the ring as well. When locating a key, the system will first find the location of the key in the consistent hashing ring by calculating the result of *hash(key)*. Then the system will travel the ring clockwise until it encounters an available node. This node will then be selected as the desired destination for the given key. When a node becomes unavailable, it will be removed from the consistent hashing ring and only the keys in the lost node need to be remapped to the next available node in the ring. A similar process will be applied when a new node joins the system. The new node will be mapped to the consistent hashing ring and the keys stored in its next clockwise available node will be recalculated and the keys whose position are at the counterclockwise direct of the newly joining node will be moved to the new node.

How Consistent Hashing Works

Consistent Hashing is based on evenly mapping each object to a point on the edge of a circle. The system maps each available node to many pseudo-randomly distributed points on the edge of the same circle. The remainder of this section will illustrate the detail of Consistent Hashing.

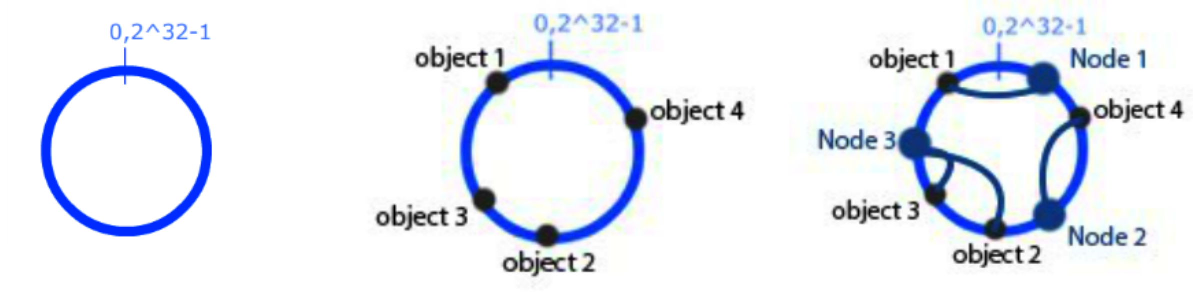


Figure 3.18: Consistent Hashing Illustration

- The Ring

  As shown in the first part of figure 3.18, consistent hashing will map a traditional hash table space with a specified table size, typically 2^{32}, to a closure ring.

- Mapping Objects to the Ring

  As shown in the middle of figure 3.18, objects will be mapped to some of the slots in the hash ring via a specially designed hash function.

- Mapping Nodes to the Ring

  The same hash function will be applied to place the actual nodes evenly on the ring. Each object will be stored in the node that is closest to it in its clockwise direction.

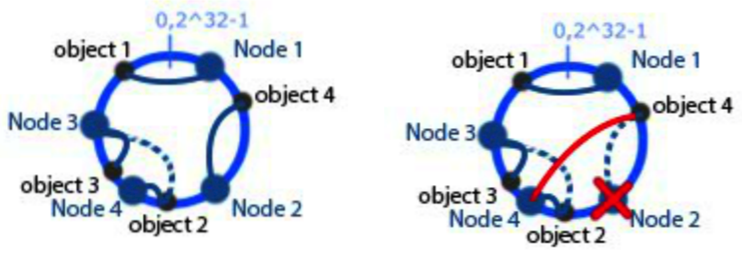


Figure 3.19: Node Addition and Removal

Figure 3.19 explains how consistent hashing deals with the addition and removal of nodes. If a new node has been added to the ring, like Node 4 in the above figure, it will first be mapped to the ring according to the same hash algorithm for mapping objects. All the objects between it and the nearest node in its counterclockwise direction will be transferred to it. Take the figure 3.19 for example, object 2 will be transferred from node 3 to node 4. When a node is removed from the ring, take Node 2 for example, all the objects it currently holds will be automatically transferred to its successor, i.e. Node 4.

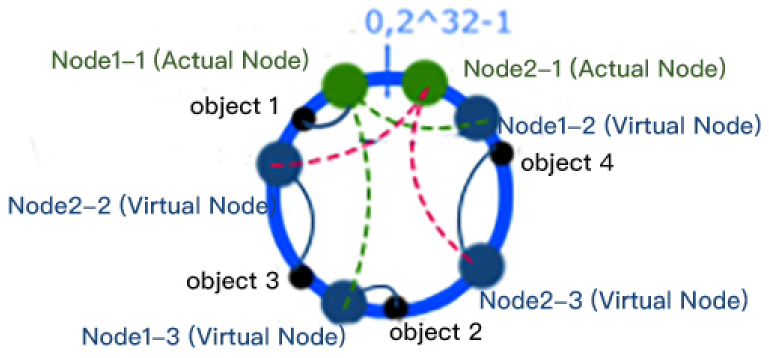


Figure 3.20: Consistent Hashing Ring Balanced by Virtual Nodes

However, when there are only a few real nodes in the hash ring, the system might be out of balance. For instance, after node 4 has been removed from the system, there will be three objects stored in node 3 while node 1 now hosts only one object, making node 3 a hot spot. Node 3 is made the hot spot because it now contains significantly more data, or ”objects”, compared to other nodes in the hashing ring. However, we do not want a hot spot because it contains an inconsistent amount of data compared to other nodes, causing the system to route a significantly higher amount of requests to the hot spot. The hot spot will not be capable of efficiently processing this massive amount of requests, making it a bottleneck within the system, while other nodes are left in idle wasting their valuable resources. This problem is solved by introducing the concept of *virtual node*. As shown in figure 3.20, an actual node will be divided into several virtual nodes. Each node, including the original node, will be labeled by its real name and an incremental suffix number. All the virtual nodes along with the actual nodes will be put together and evenly distributed to the hash ring. Therefore, the system is balanced again [68].

3.6 Summary

The goal of the literature review is to illustrate related research and technologies to develop a high-performance and fault-tolerant coordination model and framework. The following table summarizes the content of the sections in the literature review.

Section	Summary
Coordination Models and Languages	Reviews the most popular coordination model for mobile environment LIME and its predecessor Linda. The strength of LIME will remain in the new coordination model. However, the drawbacks that impact the performance of LIME will be abandoned.
Peer-to-Peer Network	Summarizes the history and concepts of P2P network. Evaluates the taxonomies of P2P architecture to figure out which kind of P2P network will better fit the proposed approach.
Connection Technology	Illustrates and compare three connection technologies: Wi-Fi, Wi-Fi Direct, and Bluetooth. The comparison reveals that the Wi-Fi network is not suitable for the proposed approach, but Wi-Fi Direct and Bluetooth seem to be ideal for implementing the new coordination framework.
REST and CoAP	The communication protocol is a very important aspect when designing a new coordination model and framework. REST is an appealing idea for the proposed approach to managing the communications between devices. This section assesses the concepts in REST and realizes that some of the features in REST are impractical for this research. It then evaluates the CoAP protocol which retains the idea of REST but is optimized for constrained devices in the mobile environment.
Distributed Hash Table and Consistent Hashing	The P2P Network section reveals that the structured P2P network will help with obtaining better data locating speed. Distributed Hash Table (DHT) is one of the most common ways for organizing a structured P2P network. Therefore, this section reviews the detail of the DHT and assesses the design of Consistent Hashing which can be leveraged to implement a high-performance and robust DHT.

Table 3.4: Summary of the Literature Review

Chapter 4

Solution and Architecture

Based on the literature review, this research develops a new coordination model that overcomes the shortages of the LIME and supports fault-tolerance as well. Moreover, it also presents a development framework that leverages the technologies mentioned in the literature review to facilitate the development of distributed mobile applications.

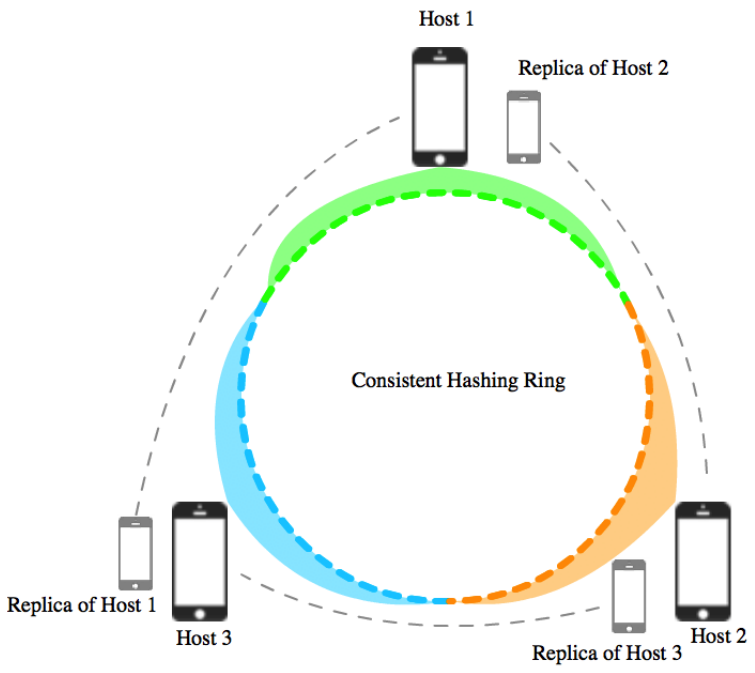


Figure 4.1: System Architecture

Figure 4.1 depicts the basic architecture of the solution proposed by this research. Each participating device is considered a host and will equally hold a portion of the data according to rules of the consistent hashing. All data is mapped to a consistent hashing ring so that the data can be efficiently managed. Each host will also store a replica of another host so that if a host leaves, then the system can restore the lost data from its replica, which enables the system to still function normally without losing any data. The remaining sections of this chapter will detail the design of the proposed approach.

4.1 Purpose and Motivation

LIME is a powerful coordination model for developing mobile applications. However, LIME has some drawbacks that severely hinders its popularity among the development of modern mobile applications, especially applications for IoT, which is currently one of the most in-demand areas.

The most significant drawback to LIME is the low performance when dealing with high-frequency engagement and disengagement. As mentioned in the LIME section of chapter 3, to ensure the integrity and accessibility of the tuples, whenever a new node is connected, LIME system needs to go over the whole federated tuple space to check whether some of the misplaced tuples can be moved to their desired destination. Similar to the engagement process, LIME system will travel the whole federated tuple space again to remove all the tuples whose target destination is the leaving node when a node disconnects from the system. Such mechanism will not be a severe problem when the system contains only a small size of participants or a small amount of data, or the network is stable enough so that engagement and disengagement will not happen at a high frequency. However, it is very common that modern mobile applications will deal with a large amount of data and the communication between nodes will be based on AD-Hoc networks which are often not stable enough. Take the IoT project mentioned in the previous chapter for example, a single node will typically contain several different sensors monitoring various arguments of the soil, and all the data needs to be exposed to other nodes for analysis purpose. Therefore, thousands of data items will be generated by a single node in a second and typically there will be hundreds of nodes placed to different areas of a farm. Moreover, engagement and disengagement might frequently happen in such system because of the following reasons:

- Some of the nodes will be driven by the battery. To reduce power consumption, those battery driven devices will only connect to the network when they want to communicate with other nodes, and the connection will be terminated once the communication is finished.

- It is impossible to provide stable WLAN coverage for the field area. The network is sustained by AD-Hoc network technologies such as Bluetooth and Wi-Fi Direct, which implies disconnection might happen unexpectedly.

Therefore, such a mechanism adopted by LIME would cause a lot of extra effort and severely impact the performance of the system when nodes frequently join and leave the network.

Another drawback of LIME is its heavyweight design of the LimeSystem ITS. LIME stores every detail of the system including the reactive statements, runtime status, and topology. A copy of the LimeSystem ITS will be distributed to each host level tuple space and all the copies need to be synchronized in an atomic step whenever the content of LimeSystem ITS is changed. The requirement of the atomic synchronization is very hard to accomplish or even impractical when the system involves a large number of participants, making the system complex and the content in the LimeSystem ITS constantly dynamic.

The third potential problem of LIME is the pattern matching mechanism it uses for finding tuples. Pattern matching is a cogent and omnipotent method for finding desired contents. Pattern matching allows LIME to place no restrictions on the form of the tuples and have anonymous tuples. Nonetheless, though powerful, LIME requires users to be extremely careful when defining the tuples. Tuples with too many elements or too complicated pattern might dramatically affect the searching performance of the system. Furthermore, most of the mobile applications do not require such unrestrained technique. Therefore, the performance of selecting tuples can be ensured by adding some restrictions to the tuples.

This paper proposes a new coordination model and designs a coordination framework for building mobile applications adapted to unstable network connections while aiming to retain the advantages of LIME as much as possible and eliminating the aforementioned disadvantages.

4.2 General Introduction

The coordination framework proposed by this paper is comprised of two parts:

- A coordination model that aims to support fast searching of data and low performance impact when dealing with engagement and disengagement. Moreover, it should be fault tolerant.
- A development framework that provides support for high-performance P2P network communication and a friendly programming interface.

The ideas of tuple and tuple space are retained. Data is still organized as tuples and stored in tuple spaces. However, after carefully analyzed two mobile projects I have participated in, I realized that tuples do not need to be anonymous in most cases. On the other hand, giving each tuple a name allows us to use the name as an index. Therefore, instead of pattern matching, we can apply plenty of sophisticated algorithms to optimize the performance for locating tuples.

As explained above, the heavy design of LimeSystemITS might drastically impact the system's performance. As such, this type of design was abandoned and a light weight global object called *SystemRuntime* is presented instead. The *SystemRuntime* will only store the topology of the current system, and a list containing the names of misplaced tuples and reactive events. Observation on the projects in which I have participated shows that for a system containing 100 nodes and over 2.3 million tuples, the size of its *SystemRuntime* is only around 0.6 MB. The whole content of *SystemRuntime* only needs to be transferred on the network once during the initialization procedure. The *SystemRuntime* object will be synchronized via operation logs after that. A single operation log only costs several bytes. Synchronizing such a small amount of data is definitely a lot easier. Furthermore, instead of using complex algorithms such as *PaxOS* to solve the consensus problem while synchronizing the *SystemRuntime*, this paper invents a mechanism which is much more simple. In the design within this paper, when a node wants to synchronize the *SystemRuntime*, it will first retrieve 3 copies of the SystemRuntime from 3 different nodes. If the system contains less than 3 nodes, the latest *SystemRuntime* will be accepted. If all of the 3 copies are exactly the same, the system will accept any of the copies instantly; otherwise if 2 copies are the same, it will accept the majority of the copies and notify the node that holds the out-of-date *SystemRuntime*. If they are all different than each other, it will try to retrieve more copies from more nodes until one copy is found the same as one of the retrieved copies. At this time all the relevant nodes will be notified to accept the copy. If no other node is available, then the *SytemRuntime* with the latest timestamp will be selected as the copy being synchronized by other nodes.

Same as LIME, tuple space is divided into 3 levels as well: Interface Tuple Space, Host Level Tuple Space, and Federated Tuple Space. Unlike LIME, once a node executes a *out*$[\lambda]$*(t)* operation, tuple t will be instantly stored in the tuple spaces and its actual location will no longer be changed no matter whether its desired destination is ready or not. Once its desired destination is ready, the destination ITS will be notified to retrieve the tuple t, otherwise tuple t's name will be stored in *SystemRuntime* until its destination becomes available.

The major improvement for coordination is done by introducing the idea of consistent hashing and replica mechanism to manage the tuple and tuple spaces. Consistent hashing can significantly reduce the time cost for finding tuples and lower the impact when participating hosts connect to/disconnect from the coordination network, while the replica mechanism can greatly help with ensuring the data integrity. Moreover, consistent hashing will also help prevent hot spots in the system. Nonetheless, this paper has customized consistent hashing to make it more suitable to the mobile environment, which will be described in detail later on.

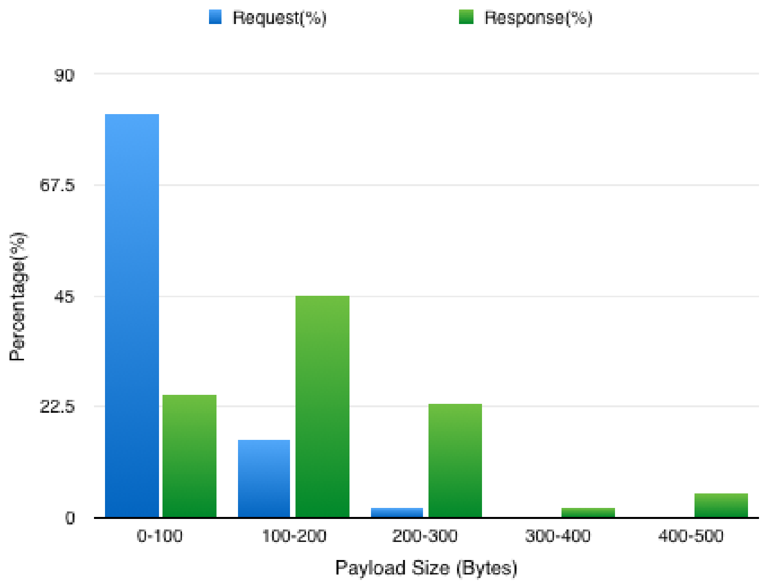


Figure 4.2: Requests and Responses Size

Besides the coordination model, this paper presents a development framework to help programmers develop distributed mobile applications more efficiently. Figure 4.2 shows the statistic result regarding the payload size of requests and responses. Sample data were collected from five different projects that all focused on developing distributed solutions running on mobile devices. Observation of the samples reveals that over 80 percent of the requests payload are less than 100 bytes and 90 percent of the responses are less than 300 bytes. Therefore, we need a lightweight protocol to facilitate highly efficient communication between nodes. CoAP seems to be a perfect choice, however, in that we assume the applications driven by the proposed coordination framework might run on unstable networks and should be able to support multiple P2P technologies such as Wi-Fi Direct and BLE, the UDP environment which traditional CoAP is relying on cannot be guaranteed. Thus this paper altered the CoAP protocol to make it compliant with different networking technologies. The UDP requirement is removed and a unified abstract network streaming model is designed to support communication in both Wi-Fi Direct and BLE network.

4.3 Coordination Model

This section will detail the design of the proposed coordination model.

4.3.1 Organization of Tuples and Tuple Spaces

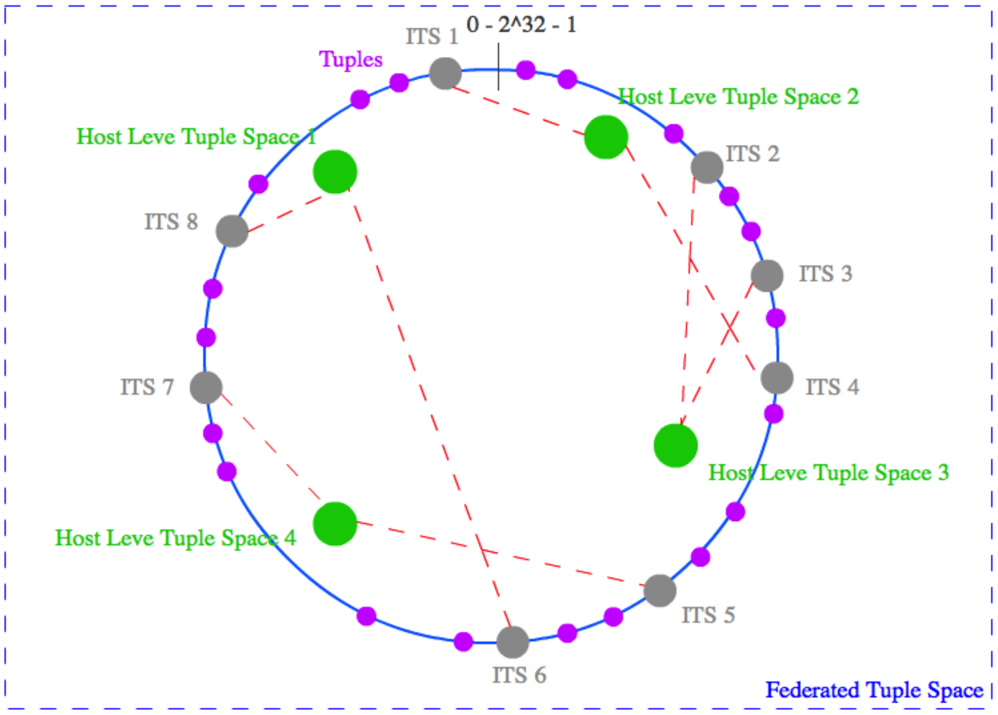


Figure 4.3: The organization of Tuples and Different Level of Tuple Spaces

Figure 4.3 depicts the organization of tuples and tuple spaces in this paper's approach. This paper utilizes the idea of *consistent hashing* to help manage the tuple spaces. All the tuples are named and mapped to the *consistent hashing ring*. Each host will be considered an actual node in consistent hashing, and the ITSs will be treated as the virtual nodes. Tuples (marked as purple dots in figure 4.3) will be stored in the node that is closest to them in their clockwise direction.

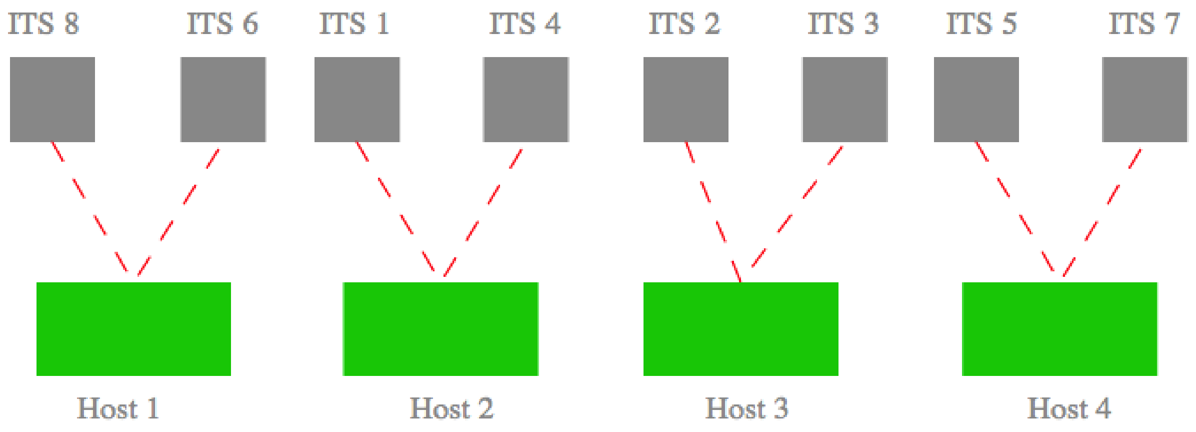


Figure 4.4: Topology of the Tuple Spaces

In traditional consistent hashing, the actual nodes and virtual nodes are mixed to participate in consistent hashing and they are identified by padding incremental numbers to the nodes' name. Each node will have the same number of virtual nodes and the virtual nodes are bonded to the corresponding actual nodes, which means when a node becomes disconnected from the network, all of its ITSs will disappear as well. Therefore, in order to enable ITSs to migrate between nodes, hosts will no longer participate in the consistent hashing procedure. Unlike LIME which allows manual migration, migration of ITSs in this paper's approach is managed by the system. The system will initially create a certain number of ITSs and evenly map them to the consistent hashing ring. Instead of mapping the host to the ring, the system will store the relationship between the hosts and the ITSs in the aforementioned *SystemRuntime*. All the relations form the *topology* of the system. As described in figure 4.4, ITS 8 and 6 belong to Host 1, ITS 1 and 4 belong to Host 2, ITS 2 and 3 belong to Host 3, and ITS 5 and 7 belong to Host 4. The system will generate enough number of ITSs during the boot up procedure to keep the system balanced.

4.3.2 Operations

All operations in LIME are maintained, but they are implemented in different ways. Locating the target of the tuple being operated is the first step in every operation, which is also the most significant difference between operations in LIME and operations in the approach described in this paper. Locating the target of the tuple contains the following steps:

1. The system will first generate a hash key by applying the hashing function to the tuple's name. Such hash key corresponds to a slot in the abstract consistent hashing ring regardless of its desired destination.

2. Then the system will travel the hashing ring in the clockwise direction until it reaches an ITS. Because each hash slot is represented by a integer, this step can be done in O(logn) (n is the number of total ITSs) by applying binary search strategy.

3. The system will turn to the topology storing in *SystemRuntime* to look up which host is holding the ITS found in step 2. The topology can be described as a hash table and the *SystemRuntime* will create a index for each ITS. Therefore, the time complexity for finding the owner of a ITS is only O(1).

When the target is found, a request containing the specific operation will be sent to the host to store it in the ITS found in step 2. If it is an output operation such as *out(t)* or *eval(t)* and it has been specified a destination ITS, step 3 will be applied to find out the host of the destination ITS. If the host is successfully found, it will be notified that there is a message for the ITS waiting to be retrieved. Otherwise, such notification will be stored as an event in the *SystemRuntime* and the host will be notified once it becomes ready. From the approach, we can see that the actual location storing the tuples might not be the same as the desired destination of the tuple. However, the system will guarantee the availability of the tuples, which will be explained in the **Fault Tolerance** section. Meanwhile, since pattern matching will no longer be used for finding tuples, parameters used for pattern matching such as current location ω and *formals* have been abandoned in this paper's design.

React operation is also supported. Every reactive statement will be considered a special tuple and be stored in the system. A pointer pointing to its real location will be registered in the *SystemRuntime*. Each reactive statement will be associated with one of the three categories: when a new host joins, when a current host leaves, and when an ITS changes (either being created, removed, or migrated from a host to another). Each reactive statement can claim itself to be one-time effective or permanent effective. Once a one-time effective reactive statement is activated, it and its pointer will be removed from the system and *SystemRuntime* immediately. Whenever an event of the above three categories happens, the system will trigger all the related reactive statements in the *SystemRuntime*.

4.3.3 Engagement and Disengagement

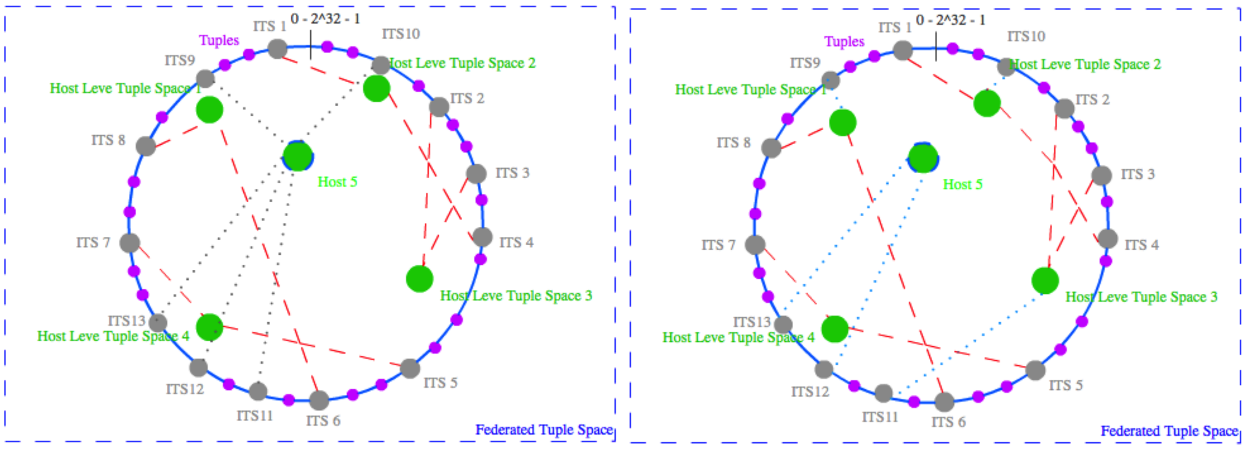


Figure 4.5: Example of Engagement

A new host can join the system with or without a set of pre-defined ITSs. The system maintains a round-robin scheduling ring to balance the ITSs among hosts to prevent hot spots. Figure 4.5 gives an example of the engagement of a new node with 5 pre-defined ITSs. Because the number of ITSs in the new host is more than the average ITSs per host of the current system, it will be re-balanced to other nodes. As a result, ITS 9 is re-balanced to Host 1, ITS 10 is moved to Host 2, ITS 11 belongs to Host 3, and the other 2 ITSs (ITS 11 and 12) remain in Host 5. After that, the system will place the new ITSs onto the ring and move designated tuples to them according to consistent hashing rules. On the other hand, if Host 5 comes without any pre-defined ITS, then some ITSs of other hosts will be migrated to Host 5. If new ITSs are brought to the consistent ring by the new host, some of the tuples might need to be moved to the new ITSs according to the rules of Consistent Hashing. For example, in figure 4.5, tuples between ITS 8 and ITS 9 will be transferred to ITS 9 from ITS 1, tuples between ITS 1 and ITS 10 will be moved to ITS 10 from ITS 2, and so forth. Since the engagement procedure only affects a small set of ITSs in the consistent hashing ring, such procedure will not drastically impact the system's performance. After all the aforementioned steps of engagement are finished, the system will turn to *SystemRuntime* to see whether there is any related *reactive statement* that can be triggered.

The procedure of disengagement is a slightly more complex than the procedure of engagement. When a host proactively leaves the system, it will broadcast a notification to all the other nodes in the network so that all the other nodes can update their local copy of the *SystemRuntime* and the neighbour nodes of the leaving node will directly proceed to the failover step which will be explained later on. Meanwhile, every host of the system will periodically send out a pulse request to every other participant to test whether they are still active. If a host detects an unexpected departure, it will launch a voting request to all other nodes. If the majority of participating nodes agree that they are no longer able to talk to the node as well, this node is marked disengaged and the system will force the participants to update their local copy of *SystemRuntime* and then proceed to the failover procedure.

A failover mechanism is designed for disengagement in order to guarantee the availability and data integrity of the system. During the failover procedure, the ITSs in the leaving host will be rebalanced to other hosts along with the tuples resting on it. The round-robin scheduling ring will guarantee the balance of the system by always migrating a ITS to a host sustaining least ITSs in the system. The system will go back to normal once the failover procedure is done.

4.3.4 Fault Tolerance

As mentioned in previous sections, the system involves a *fault tolerance* design to guarantee the availability of tuples and ITSs. Moreover, *fault tolerance* is also the key component for implementing the failover mechanism. This paper achieves the goal of being fault tolerant by storing a replica for each host in another host. For example in figure 4.6, host 2 holds a replica of host 1, host 3 holds a replica of host 2.

When a new node joins the network, the replicas of affected hosts need to be regenerated and redistributed after the re-balancing procedure is finished. Such a procedure will be done in the background, so as to not

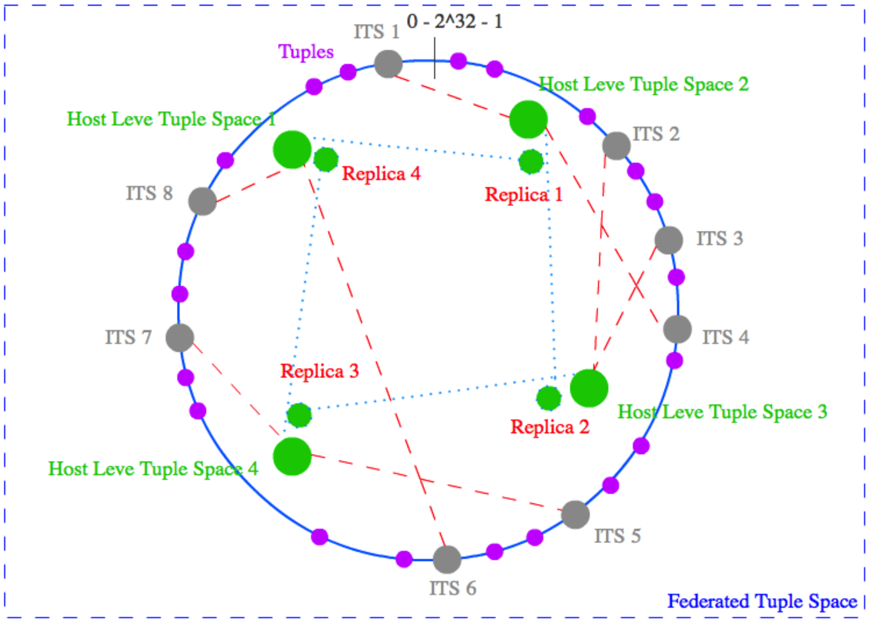


Figure 4.6: Example of Replica Mechanism

impact the system's performance.

When a host leaves the network, all the ITSs resting on it will disappear immediately. Thus the system will enter the failover state. In the failover state, the system will redistribute the ITSs from the leaving node's replica to other hosts. Once failover is done, the system will be back to work and all involved nodes will regenerate their back-ups and re-distributed among hosts, which is the same as the last step in engagement.

Because during the procedure of generating and synchronizing the replicas, tuples in the affected ITSs might be changed (new tuples might be generated, current tuples might be modified or deleted), the system will record the operations log and eventually synchronize the changes to corresponding replicas.

Chapter 5

Implementation

5.1 Design

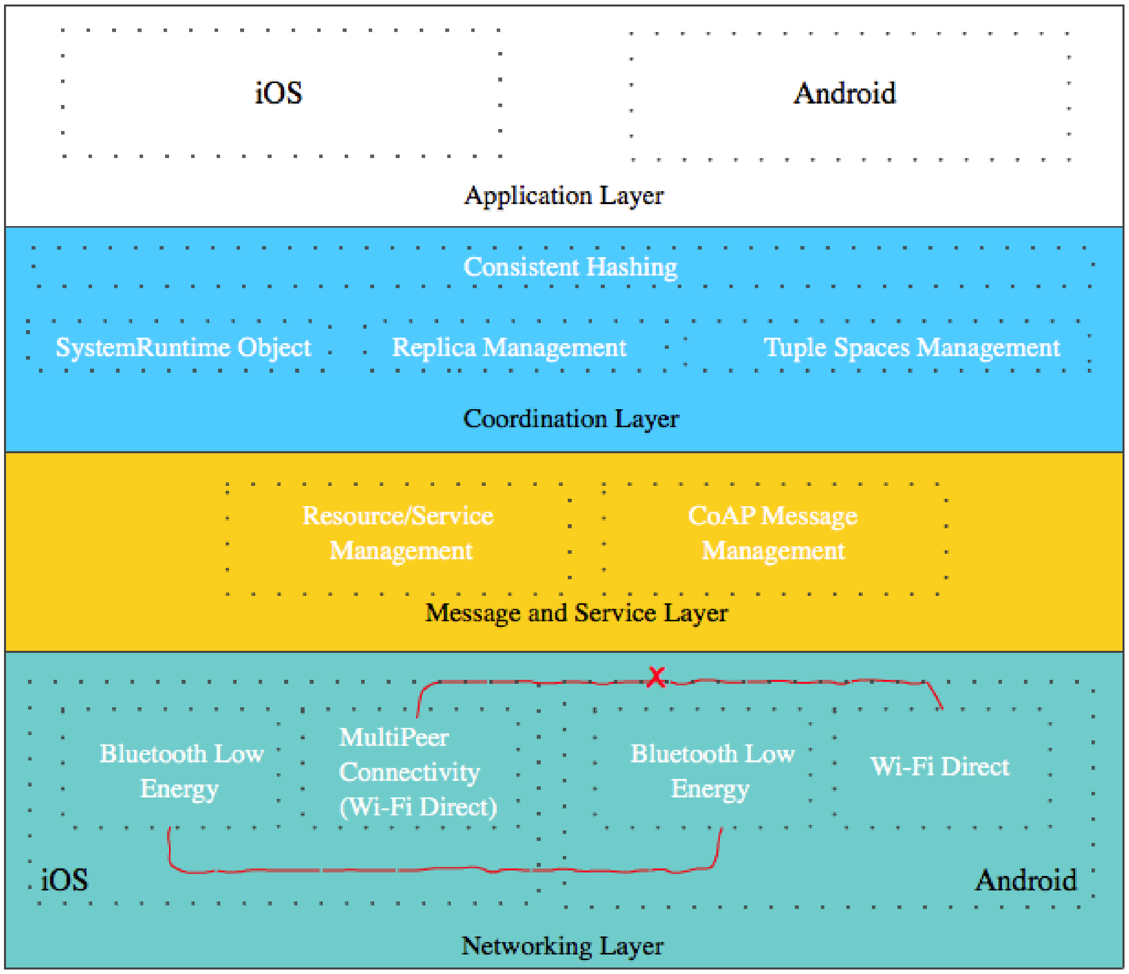


Figure 5.1: Architecture of the Coordination Framework

This paper presents a cross-platform coordination framework for the mobile environment. With the help of the framework, programmers can easily develop and build distributed mobile applications running on major platforms such as iOS and Android. This framework involves *Xamarin* as the development environment to provide cross-platform support. *Xamarin* comes with a software developing tools set which uses C# as the intermedia programming language and allows developers to write native Android, iOS, and Windows Phone apps with native user interfaces. Another important feature of *Xamarin* is that it allows developers to share code across multiple platforms so that it can dramatically reduce the time cost of developing mobile apps.

Figure 5.1 depicts the architecture of the coordination framework. The top first layer is the application layer where developers use native API to present the user interface and build the functionalities of their apps. The other three layers constitute the core of the framework. The coordination layer consists of four modules: consistent hashing, SystemRuntime object, replica management, and tuple spaces management. The Consistent Hashing module routes each request to the correct host and keeps the workload of each host balanced. SystemRuntime Object component will be responsible for managing the aforementioned global SystemRuntime. The replica management component and the Tuple spaces management component will manage the replica and tuple spaces respectively. The Message and Service Layer contains two component and is responsible for managing the service provided by the hosting node, and the incoming and outgoing CoAP messages. The Service management component analyzes all the incoming requests and distributes it to corresponding resources, returning the output to the node which sent out the request. It also provides interfaces for app developers to register their customized services. The CoAP Message management component packs all the outgoing messages and unpacks all of the incoming messages. Errors or exceptions will be handled by the CoAP Message management component as well. The bottom layer is the Networking Layer that performs the actual data transmission and physical connection. The Networking Layer supports BLE and Wi-Fi Direct communication, however, only BLE can be used for cross-platform communication because iOS and Android do not follow the same Wi-Fi Direct standard.

Figure 5.2 depicts how the layers are organized in the framework. There is a key component called the *Framework Core* that ”stands” in the middle coordinating all the layers. When the network layer receives a raw CoAP message in binary format, the CoAP Messages component will unpack it to a CoAPRequest object and dispatch it to the Framework Core. Threads in the application layer can also interact with the framework by making a specific function call which will eventually be processed by a worker thread in the framework. The messages will be processed by the Framework Core through several steps and then the processed result (either a CoAPRequest or a CoAPResponse) will be passed back to the network layer to actually send to relevant nodes in the system. The message/service layer and the coordination layer are involved during the message processing procedure.

The remaining sections of this chapter will cover the detailed explanation of each layer in the framework.

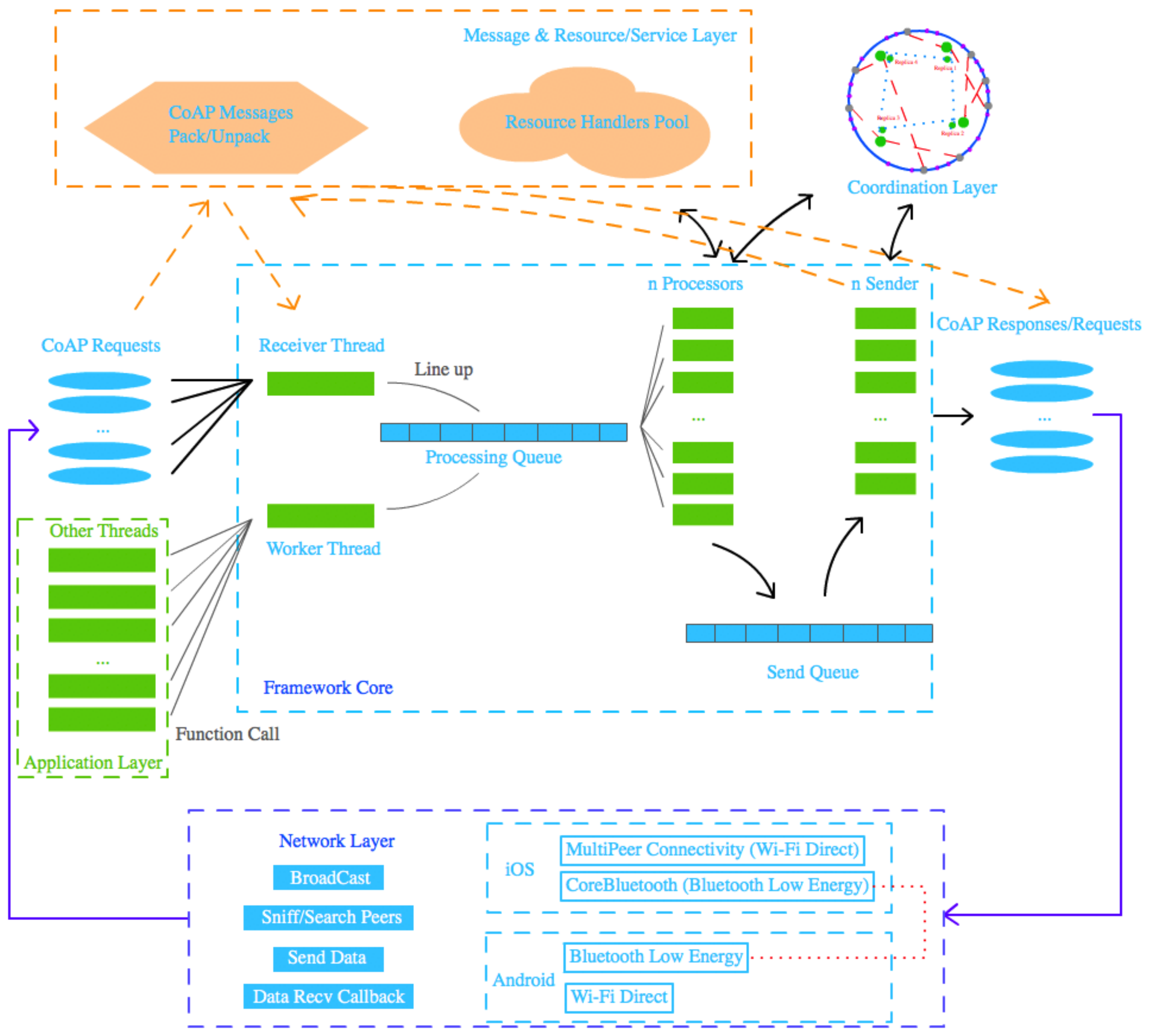


Figure 5.2: Relation between Layers

5.2 Coordination Layer

5.2.1 Consistent Hashing

Consistent hashing is the key component for performing the coordination because it calculates the position of the tuples and ITSs on the consistent hashing ring.

Figure 5.3 depicts the design of the consistent hashing component. It mainly consists of 5 classes: Ring, FNVHash, ITS, ITSList, and ITSLocator. *Ring* represents the consistent hashing ring. *FNVHash* defines the hash function that will be used by *ITSList* to map the ITSs to specific slots on the *Ring*. All *ITSs* are stored in an *ITSList*.

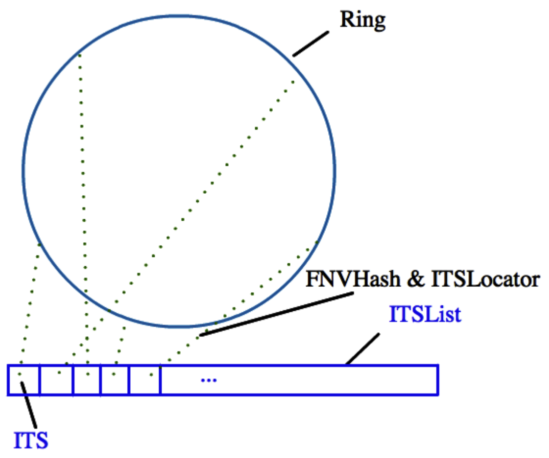


Figure 5.3: Implementation of the Consistent Hashing

The consistent hashing ring is defined as a template class to support multiple types of elements and it is directly inherited from *SortedDictionary<uint, T>* so that we can apply binary search to accelerate the speed of locating a tuple or ITS on the ring. Each element on the Ring is indexed by a hash value which is calculated by FNVHash.

```
public class ITS {
      public string Name{get;set;}
      public uint        Index{get;set;}
      public string Host{get;set;}
}
public abstract class ITSList {
        private Ring<ITS> Circle = new Ring<ITS>();

        void Initialize(ITS[] ITSs )  {
           lock (syncLock) {
                for (int i = 0; i < ITSs.Length; i++) {
                          uint hash = FNVHash.Hash(ITSs[i].Name + "-" +
                              ITSs[i].Index.ToString());
                          Circle[hash] = ITSs[i];
                }
                Keys = Circle.Keys.ToArray();
                this.IsInitialized = true;
                nodes.AddRange(ITSs);
           }
       }
        void Add(ITS its)  {
                uint hash =FNVHash.Hash(its.Name + "-"+ its.Index.ToString());
                Circle[hash] = its;
                nodes.Add(its);
       }
```

```
        ITS Remove(ITS its) {
                        string key = its.Name + "-" + its.Index.ToString();
                uint hash =  FNVHash.Hash(key);
                Circle.Remove(hash);
                Keys = Circle.Keys.ToArray();
                return ITSLocator.locate(Circle , key);
        }

        /**
         *Implementation for other functionalities*
         **/
}
```

As defined in the above code, each node on the ring represents an ITS. During the boot-up period, the framework will first prepare a specific number of empty ITSs according to the configuration, or move some of the ITSs from other hosts if needed. Each ITS has three elements: the unchangeable Name, Index, and a string indicating which host it is resting on. The name and index of a ITS will be used to generate the hash value of the ITS and such hash value determines the ITS's position on the ring.

The framework will then retrieve the basic information of all the ITSs resting on other hosts by synchronizing the SystemRuntime Object. After that, the framework will reproduce the consistent hash ring of the current system by calling *ITSList.Initialize()* and passing all the ITSs as the parameter of the function call.

During engagement and disengagement, the framework will call *ITSList.Add()* and *ITSList.Remove()* respectively to notify the consistent hashing component about the change. Specifically, when disengagement happens, the *ITSList.Remove()* function will return the next available ITS in the system so that the Replica Management component knows how to perform failover to guarantee the data integrity.

5.2.2 The SystemRuntime Object

The SystemRuntime object is a global object containing two members: topology and eventlist, which are defined in the following code snippet:

```
public sealed class SystemRuntime {
        public static SystemRuntime Instance {
            get {
                if (instance == null) {
                    lock (oplock) {
                        if (instance == null) {
                            instance = new SystemRuntime ();
                        }
                    }
                }
                return instance;
            }
        }
        private SystemRuntime () {
            topology = new Dictionary<string, List< ITS > > ();
            eventlist = new Dictionary<string, List< Event_t >> ();
            sync_thread = new Thread (SyncSystemRuntimeObj);
            sync_thread.Start ();
        }


        private static SystemRuntime instance = null;
        private static readonly object oplock = new object();

        private Dictionary<string, List< ITS > > topology;
        private List<Event_t> eventlist;
        private Thread sync_thread;

        /**
        *Implementation for other functionalities*
        **/
    }
```

As shown in the above code snippet, the topology is an array that stores all of the connected devices and describes the relationship between the hosts and the ITSs. The array is indexed by the name of the host and its element is an array of the ITSs resting on the associated host. The eventlist contains the list of events that are waiting to be processed. The structure Event_t is defined as follows:

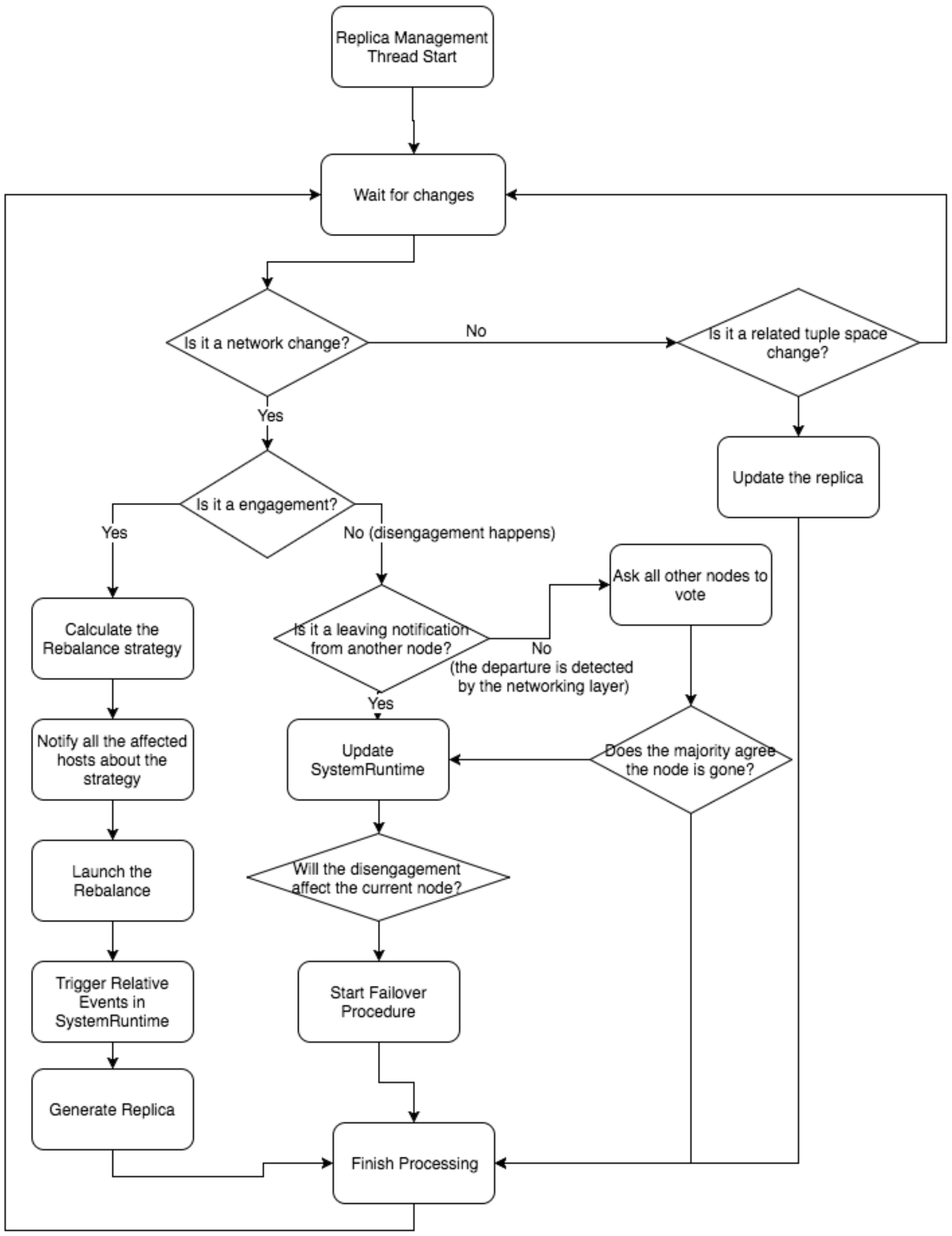
```
public sealed class Event_t {
        public string hostname;
        public ITS its;
        public string evdata;
}
```

In the current implementation, the global SystemRuntime object is designed under the idea of the *Singleton Pattern*. It will be automatically created during the first time it was called. Once it's created, it will launch a thread called SyncSystemRuntimeObj to synchronize the object with other connected nodes. The SyncSystemRuntimeObj will periodically serialize the SystemRuntime object to a binary string and then synchronize the binary string with other involved nodes following the rules described in the previous section. If the binary string it received from other nodes is the same as the one it has, this means its SystemRuntime is the latest version and it doesn't need to do anything. Otherwise, it will deserialize the received binary string and update its SystemRuntime object.

Because the framework is supposed to run on many constrained devices with limited power supply and hardware resources, the implementation has been optimized for better performance in many subtle details. For example, the previous code snippet uses the *double checked locking* mechanism to avoid unnecessary lock while retrieving the instance of SystemRuntime.

5.2.3 Replica Management

The replica management component is the key element for implementing the fault tolerance feature and guaranteeing the data integrity of the system.



Replica Management Thread Start
Wait for changes
Is it a network change?
No
Is it a related tuple space change?
Yes
Update the replica
Is it a engagement?
Yes
No (disengagement happens)
Ask all other nodes to vote
Calculate the Rebalance strategy
Is it a leaving notification from another node?
No (the departure is detected by the networking layer)
Yes
Notify all the affected hosts about the strategy
Update SystemRuntime
Does the majority agree the node is gone?
Launch the Rebalance
Will the disengagement affect the current node?
Trigger Relative Events in SystemRuntime
Start Failover Procedure
Generate Replica
Finish Processing

Figure 5.4: Flowchart of the Replica Management Component

Figure 5.4 depicts the detail of the replica management implementation. The replica management thread will be launched when booting up the framework. After the replica management thread finishes initializing itself, it will hang up until it receives notification from the framework. Whenever the framework detects a network change or a tuple space change that affects the current node, it will notify the replica management thread and wake it up. Once it receives the signal, the thread will first test whether it relates to a network changing event. If yes, this means either a new host joined the system or an existing node left the system. Otherwise, it will measure whether such event contains a change that might affect the tuple spaces storing in the host. If yes, it will start a necessary process to ensure the system's data integrity. Conversely, if such changes do not affect the current host, the replica management thread will go back to hang-up mode and wait for further notification.

Engagement

If a new node joins the system, as described in the previous section, the replica management thread will first calculate the rebalance strategy according to the situation. For instance, if the number of ITSs being carried by the new node is more than the average, some of the ITSs in the new node will be migrated to other nodes following the aforementioned round-robin scheduling ring schema. The thread will generate a strategy describing how to balance the system after involving the new ITSs, then send the strategy to all affected nodes and start the rebalancing procedure. The thread will notify the SystemRuntime component to start synchronization afterward. Once the synchronization is finished, events related to the ITSs resting on the current host will be triggered.

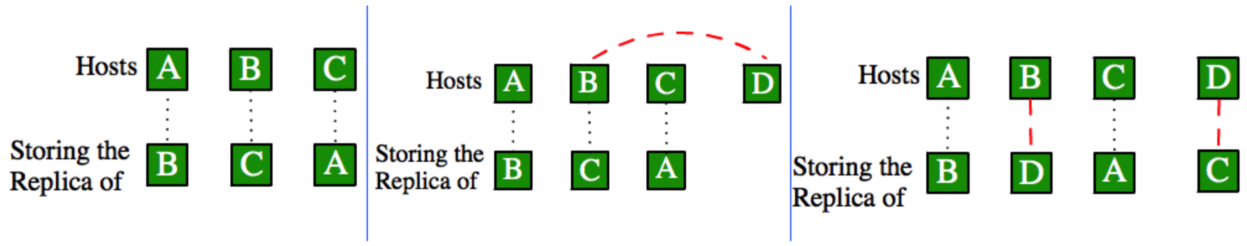


Figure 5.5: Replica Mapping

The last step for engagement procedure is to generate a replica of the current host in another node. As shown in the left-most part of figure 5.5, the replica management component maintains an array that stores the mapping rules of the original data and their replicas. For the example in figure 5.5, host A is storing the replica of host B, the replica of host C is storing in host B, and host C is holding host A's replica. The system can be considered in *healthy status* only when every participating host has a replica storing in another host. As the middle part of figure 5.5 reviews, when host D joins the system, the replica management component of host D will first randomly pick a host from the array to store the replica of current host, i.e. host B. Then host D will move the replica of host C from host B to itself, and store a replica of itself in host B. Hence, the system can be in *healthy status* again.

Disengagement

If the framework detects that a node is leaving the system, it will notify the replica management thread to intervene. The departure of a node can be detected in 2 ways: either the departure node proactively broadcasts a leaving notification or one of the nodes realizes that it can no longer talk to the leaving node. If a node receives a leaving notification, it can directly proceed to the next step. However, in most cases, disengagement is caused by unexpected issues such as a node is out of power, or it's been taken out of the effective range. In such case, the leaving node will not be able to have the opportunity to proactively send out the leaving notification so the departure has to be detected by another active node. However, when a node realizes it cannot reach another node, it should not instantly mark the node as ”departed” since the failure might be caused by temporary issues such as network traffic, putting it in a state where it can be resumed at a later time. Therefore, instead of marking the node as ”departed” directly, the node which has detected the disconnection will mark the node as ”potentially departed” and broadcast a vote request to all other active nodes in the system to vote whether they can still talk to the ”potentially departed” node. If the majority of the nodes agree that they can not see the ”potentially departed” node anymore, the originating node will mark the ”potential departed” node as ”departed” and it will formally announce it to all other nodes so that all the nodes, including itself, can proceed to the next step of the disengagement procedure.

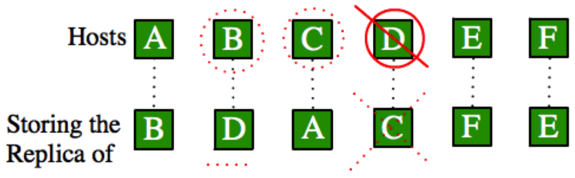


Figure 5.6: Disengagement Example

The next step of the disengagement is the failover procedure mentioned in the coordination model section of chapter architecture. First, the node will figure out whether it needs to be involved in the failover. If not, the node can go back to normal state directly. Take figure 5.6 for example, host D is leaving the system and only host B and C should be involved in the failover phase because host B is holding the replica of host D and the replica of host C is lost. So host A, E, and F can immediately go back to normal and host B will start the failover procedure.

Host B starts the failover by retrieving all the ITSs in D from the replica, and takes all the ITSs stored within it and spreads them over the existing hosts following the guidance of the round-robin scheduling ring to balance the system and prevent a hot-spot. After that, just like the example in figure 5.5 where D creates its replica, host C will randomly pick a host to store its replica and move the replica in that host to itself. Be aware that during the period of rebalancing, the tuple spaces or other nodes who are assigned with new ITSs will be changed, therefore they will need to update their replicas as well.

Tuple Spaces Change

When the tuple spaces of a node change, the node needs to update its replica. Such change includes the situation such as some ITSs are moved to this node, some ITSs are rebalanced to other nodes, or tuples stored in its ITSs are created or modified by executing *out* or *eval* operations.

5.2.4 Tuple Spaces Management

Tuple Spaces management is mainly responsible for managing the tuple spaces of a node and performing operations on the tuples stored in those tuple spaces.

```
public class Tuple {
        public string Name { get; set; }
        public string Value { get; set; }
        public Dictionary<string, string> Values { get; set; }
}
```

Above code is the primary definition of a tuple. It has a name attribute and a value attribute. It is sufficient in most cases, however, sometimes a tuple needs to have more than one field. Therefore, the framework involves a 2D array which is indexed by the field's name.

```
Dictionary< string /*ITS name*/, Dictionary< string /*tuple name*/, Tuple> > TupleSpaces;
```

All the tuples that belong to a node are stored in a data structure called *TupleSpaces* which is defined in the above code. Each element of the *TupleSpaces* is an array that stores all the tuples of an ITS which is indexed by the ITS's name. Each of the arrays represents an ITS and it uses the tuples' name to index the tuples. A thread will be launched to manage the *TupleSpaces* and perform operations on the tuples. The consistent hashing component will route all the operation requests related to the current node to the thread.

5.3 Message and Service Layer

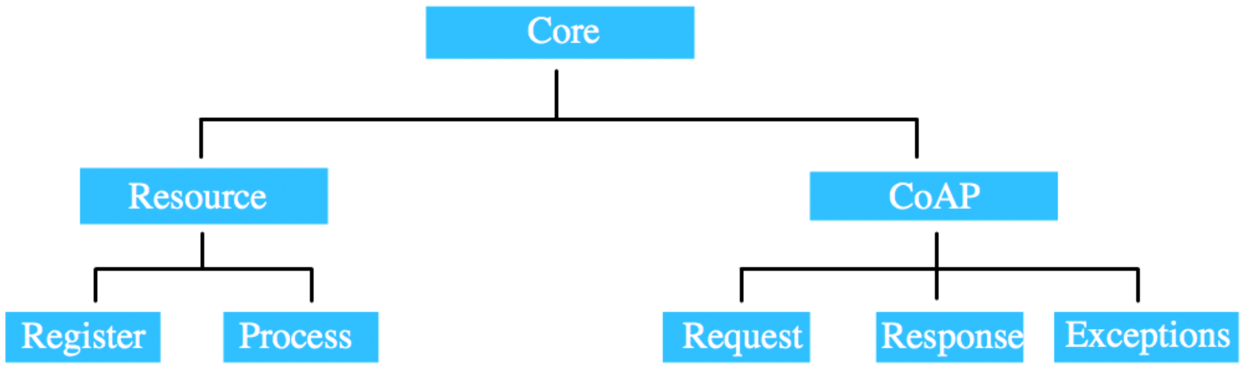


Figure 5.7: Organization of Message and Service Layer

Figure 5.7 illustrates the organization of the Message and Service Layer. The Core class will create a number of *processor* threads to be responsible for coordinating the Resource class and the CoAP class to work together properly. When the CoAP module unpacks a CoAP message targeting a specific resource, it will be placed into the queue that stores all incoming messages, and one of the *processor* threads created by the Core class will ask the Resource module to provide the corresponding handler to process the message.

5.3.1 Service/Resource Management

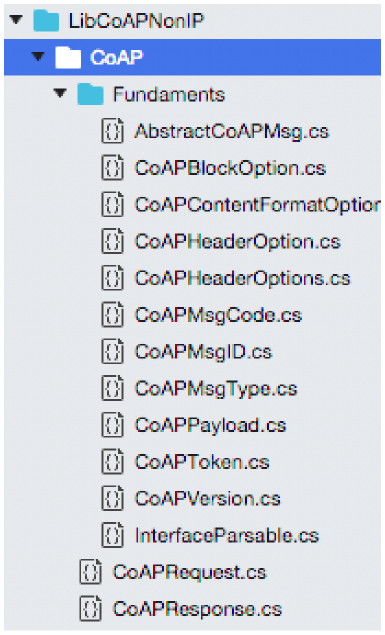
In this framework, resource is a synonym of service. Resource is a term defined in CoAP standard, thus in order to be compatible with CoAP standard, the framework uses the term resource instead of service.

```
public delegate CoAPResponse RequestHandler(Device sender,CoAPRequest request);
public class Resource {
        public Resource(string Name, RequestHandler Handler);
        public RequestHandler GetHandler();
        public CoAPResponse ProcessRequest();

        private Resource();
}
public class FrameworkCore {
        public void RegisterResource(string name, RequestHandler handler) ;
        public void DeregisterResource(string name);

        private Dictionary< string , Resource > resources_pool;

        /**
         * other definition...
         */
}
```

The above code depicts the key elements implementing the service/resource module. As shown in the definition of class *Resource*, the user must specify a handler to process related requests when creating a new resource. The request handler is a delegate (i.e. a function pointer) that takes the sender and original request as the input parameters and returns a CoAP response for the request. A user can register and deregister a resource by making corresponding function calls defined in the FrameworkCore class. The Framework Core maintains all the registered resources in a member called resources_pool. Such mechanism enables the user to customize the behavior of the app.

5.3.2 CoAP Message Management

The CoAP standard requires all the messages being transferred under UDP protocol. Using UDP implies the presence of an IP address. However, many ad-hoc P2P networks such as Bluetooth or Wi-Fi Direct in iOS do not support IP protocol. Therefore, the framework has to develop its own CoAP implementation to make it compatible with the Non-IP based network environment.



```
LibCoAPNonIP
  CoAP
    Fundaments
      AbstractCoAPMsg.cs
      CoAPBlockOption.cs
      CoAPContentFormatOptior
      CoAPHeaderOption.cs
      CoAPHeaderOptions.cs
      CoAPMsgCode.cs
      CoAPMsgID.cs
      CoAPMsgType.cs
      CoAPPayload.cs
      CoAPToken.cs
      CoAPVersion.cs
      InterfaceParsable.cs
    CoAPRequest.cs
    CoAPResponse.cs
```

Figure 5.8: Organization of the CoAP Implementation

Figure 5.8 is a screenshot of the CoAP implementation. It reveals that the major interfaces for interacting with other components are the CoAPRequest class and the CoAPResponse class. They are both inherited from the base class AbstractCoAPMsg which defines how to serialize a CoAP message to a binary string and how to deserialize CoAP message from a binary string. This is the most important change compared to the original CoAP standard. The original CoAP standard will directly operate a message on the network using UDP protocol. In this framework's implementation, the CoAP implementation will only interact with the binary stream while the actual network transmission will be done by the network layer. This not only clarifies the division but also makes it possible for the Non-IP environment to support CoAP communication. All other requirements defined in the CoAP standard have been strictly followed.

5.4 Networking Layer

As depicted in the network layer part of figure 5.2, the network layer of the framework defines the required interfaces regardless of the platform and the communication technology. The basic required interfaces are: *Broadcast*, *Sniff/Search Peers*, *Send Data*, and *Data Recv Callback*.

- The *Broadcast* interface requires the framework to be able to either constantly or periodically broadcast the network signal so that other peers can discover the P2P network and request to join.

- The *Sniff/Search Peers* interface demands that a node should be able to discover other nodes by sniffing the network signal. The difference between *Sniff* and *Search* is that sniffing will only try to discover other nearby nodes, while searching will also try to connect once a node is discovered.

- The *Send Data* interface allows users to transmit streaming data among the network.
- As described in the **resource management section**, a user should be able to customize the app's behavior when receiving data from other nodes. The framework accomplishes this by allowing the user to register self-defined callback functions to the network layer. The callback functions will be automatically triggered once new data is incoming.

The network layer supports both iOS and Android platforms, and is compatible with the Wi-Fi Direct and BLE technology.

5.4.1 Wi-Fi Direct

The proposed coordination framework implements Wi-Fi Direct communication on both iOS and Android platform.

Implementation on iOS

iOS does not provide API to directly access the Wi-Fi Direct features. Therefore, using the *MultiPeerConnectivity* API is the only way to support Wi-Fi Direct communication in iOS. *MultiPeerConnectivity* API involves multiple communication technologies including Wi-Fi Direct, Wi-Fi, and Bluetooth. The system may choose to use different technologies for communication depending on the scenario. Therefore, the performance of *MultiPeerConnectivity* varies from time to time. Moreover, it is also the reason for not being able to communicate with Android Wi-Fi Direct because the *MultiPeerConnectivity* APIs do not comply with the Wi-Fi Direct standard.

```
public class PeersNetwork {
        private MCSession rr_mySession;
        private MCPeerID rr_myPeerID;
        private Device rr_myDevice;
        private MCNearbyServiceAdvertiser rr_broadcaster;
        private MCNearbyServiceBrowser rr_seeker;

        public override int SendData(Device[] Destination, byte[] payload) {
            if (payload == null || payload.Length == 0) {
                Console.WriteLine("No payload");
                return -1;
            }
            rr_oplock_servers.AcquireReaderLock(-1);
            for (int i = 0; i != Destination.Length; ++i) {
                if (!rr_activeServers.ContainsKey(Destination[i].DisplayName)) {
                    Console.WriteLine("Device Lost");
                    Destination[i].PeerID = null;
                } else {
                    Destination[i].PeerID =
                        rr_activeServers[Destination[i].DisplayName].PeerID;
                }
            }
            rr_oplock_servers.ReleaseReaderLock();
            rr_oplock_sendQueue.AcquireWriterLock(-1);
            rr_sendQueue.Add(new SendQueueElement(Destination, payload));
            rr_oplock_sendQueue.ReleaseWriterLock();
            for (int i = 0; i != Destination.Length; ++i) {
                if (Destination[i].PeerID != null) {
                    rr_DataThreads[Destination[i].DisplayName].Signal.Set();
                }
            }
            //notify data threads to send out the data
            return payload.Length;
        }

        /**
         * Other definition
         */
}

public class SessionDelegate : MCSessionDelegate {
        public override void DidReceiveStream(MCSession session, NSInputStream stream,
            string streamName, MCPeerID peerID) {
            /**
             * Other definition
             */
            if (rr_caller.DataRecvFunc == null) {
                throw new NetworkException("DataRecvFunc not set");
            }
            rr_caller.DataRecvFunc(client_dev, data);
        }

}
```

As shown in the code snippet above, a node will use a *MCNearbyServiceAdvertiser* object to constantly broadcast the network signal. Sniffing and Searching will be done by a *MCNearbyServiceBrowser* object. All the context will be saved in a *MCSession* object and peers are identified by *MCPeerID* and *Device*. *SendData* function call will save the data to be sent in a send queue. Later on, the data will be dispatched by the actual data transmitting thread. The locking mechanism has been widely used to guarantee the thread safety and the Reader/Writer lock is used to improve the performance. Moreover, the data transmitting thread uses streaming API to obtain faster data transmitting speed. Callback function for receiving data will be registered in the *SessionDelegate* of the *MCSession* object and it will be triggered at the end of the data stream receiving function.

Implementation on Android

Instead of having a *MCNearbyServiceAdvertiser* and *MCNearbyServiceBrowser* separately, Android uses a single *WifiP2pManager* to process the tasks. Although Android does not share the same API with iOS, the concept is relatively the same. The main differences are the APIs' name and their corresponding parameters.

5.4.2 Bluetooth Low Energy

Because both iOS and Android supports the same standard of BLE, cross-platform communication is possible via the BLE technology. For this reason, this section will only use iOS for examples. Broadcasting is done by a *PeripheralManager* object. The *PeripheralManager* will broadcast the services and characteristics to the nearby area. A *CentralManager* of another device will search for existing peripherals. Similar to Wi-Fi Direct, each node will be broadcasting signal while also searching for other nodes. That is to say, each node will be undertaking the roles of peripheral and central at the same time. *Send Data* is a bit more complicated in BLE. It involves two steps: In the first step, the sender will use its central role to post a read request to the peripheral role of the target node along with the designated characteristic for sending the data. The second step occurs when the target node will use its central role to subscribe the given characteristic to obtain the data. Data Recv Callback function will be called when new data is received.

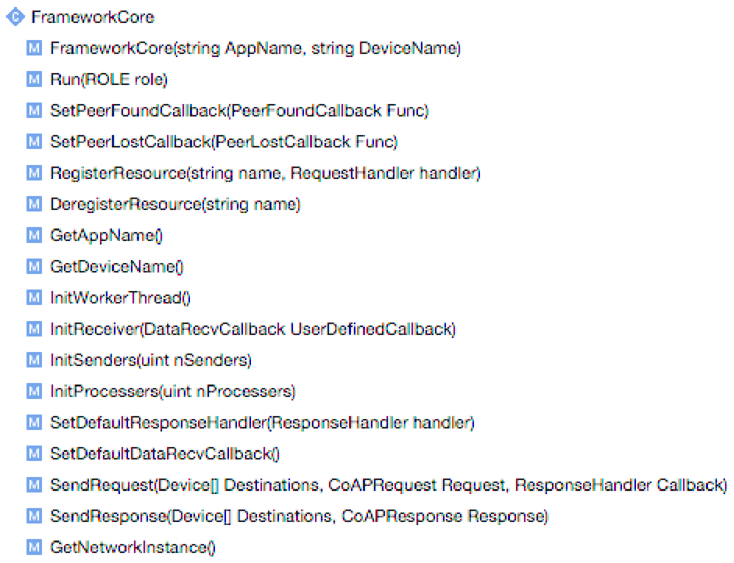
5.5 The Framework Core

The Framework Core is the pivotal element to interact with the users and coordinates each part of the framework.

Figure 5.9 lists all the key member functions of the FrameworkCore class. During the boot up period, the framework will first initialize the worker thread, receiver thread, processors, and senders. After that, the framework will register the resources predefined by the user and initialize the network as well.

Once the framework is booted up, the FrameworkCore will first try to seek other existing nodes for 5 seconds. If no other node is found, it will try to create its own cluster by broadcasting the network signal to the nearby area. Otherwise, it will request to join the discovered cluster.

As depicted in figure 5.2, the Framework Core has a receiver thread to accept all the incoming CoAP requests. The framework uses message queues for inter-process communication. Therefore, the receiver thread will line up all the incoming requests into a processing queue for the processors to retrieve. Meanwhile, threads defined by the user can also pass messages to the Framework Core by making a function call to the predefined worker thread in the Framework Core. The worker thread will also line up the messages into the processing



```
FrameworkCore
  FrameworkCore(string AppName, string DeviceName)
  Run(ROLE role)
  SetPeerFoundCallback(PeerFoundCallback Func)
  SetPeerLostCallback(PeerLostCallback Func)
  RegisterResource(string name, RequestHandler handler)
  DeregisterResource(string name)
  GetAppName()
  GetDeviceName()
  InitWorkerThread()
  InitReceiver(DataRecvCallback UserDefinedCallback)
  InitSenders(uint nSenders)
  InitProcessers(uint nProcessers)
  SetDefaultResponseHandler(ResponseHandler handler)
  SetDefaultDataRecvCallback()
  SendRequest(Device[] Destinations, CoAPRequest Request, ResponseHandler Callback)
  SendResponse(Device[] Destinations, CoAPResponse Response)
  GetNetworkInstance()
```

Figure 5.9: Key Member Functions of FrameworkCore

queue with a flag so that the processors can know it is not a CoAP request but a message from a local thread. There will be many processors running at the same time to improve the performance. Each processor will retrieve one message at a time from the processing queue. Depending on the message's content, the processor might need to cooperate with the coordination layer to retrieve the correct destination of the message. For example, a message containing an *out(t)* operation will need to know the destination of tuple *t*. Once a message is processed, the result will be saved into a send queue. Multiple senders will continuously pop messages from the queue and send the message to corresponding nodes. If the destination of a message is unknown, the sender will refer to the coordination layer for the answer. After the messages have been packed into binary streams by the CoAP Messages Component, the sender will dispatch the messages to its desired destination via the network layer.

Chapter 6

Experiments

This chapter details the experiments on evaluating the performance of the coordination framework. The purposes of the experiments are:

1. Bandwidth Test

   This experiment tests the raw data rate of the network without specifying any payload to the network packages. This experiment will indicate the upper limit of the network capacity and performance.

2. Basic Throughput

   This experiment tests the throughput between two nodes without the intervention of the coordination model. Tests with different size of payload will be repeated at least 1,000 times so that abnormal result can be filtered out. The result of this test along with the result of other experiments will be used to analyze the performance of the proposed coordination model.

3. Throughput of the Coordination Framework

   The purpose of this experiment is to retrieve the throughput between two nodes with the intervention of the coordination model. Same as the previous experiment, each test of this experiment will be repeated at least 1,000 times to filter out abnormal data. If the result is close to the result of the basic throughput experiment, it means the proposed coordination model is a high-performance model because its intervention does not notably affect the network throughput.

4. Engagement and Disengagement

   This experiment has two purposes:

   (a) Reveal the performance of the coordination model while dealing with engagements and disengagements.

   (b) Verify whether the proposed coordination model is fault-tolerant or not.

5. Data Distribution

   As mentioned in the consistent hashing section of the literature review, hot spots can severely affect the performance of the system. This experiment is set up to prove that the proposed coordination model can evenly distribute the data to all involving nodes and it will not generate a hot spot.

6.1 Experiments Environment and Bandwidth Test

Device	Units	Operating System
iPad Air 2 (16G)	2	iOS 8.4
iPhone 6 (64G)	1	iOS 8.4
iPod Touch (32G)	4	iOS 8.4

Table 6.1: Devices Configuration

Seven iOS devices were involved in the experiments. The detailed configuration of the seven devices is listed in table 6.1. All of the devices supported both Wi-Fi Direct and BLE. Because there is no explicit API for Wi-Fi Direct functionalities in iOS, when testing performance under Wi-Fi Direct, all the involved devices were disconnected from all Wi-Fi networks and their Bluetooth modules were completely turned off in order to force the system to use the Wi-Fi Direct channel.

The initial experiment tries to reveal the data rate of Wi-Fi Direct and BLE. During the experiment, a device continuously sent streaming data to the other device. The receiving device calculated how much data it received every second and used this value to calculate the data rate. The data rate was tested under different cases such as two iPads, an iPad and an iPhone, an iPad and an iPod touch, an iPod touch and an iPhone. All the results are similar to each other. Therefore, this paper only shows the result of the data rate between two iPads here as a representation.

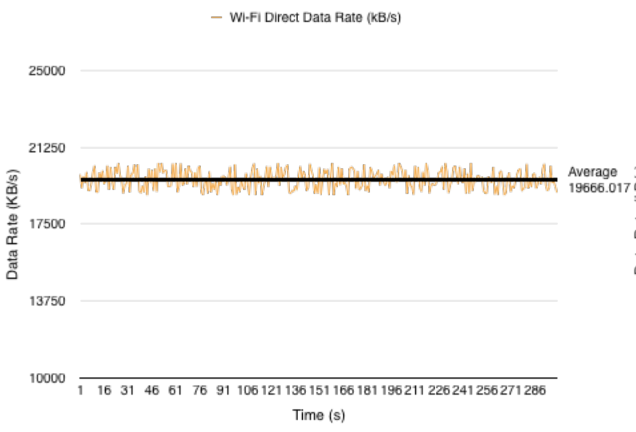


Figure 6.1: Wi-Fi Direct Data Rate

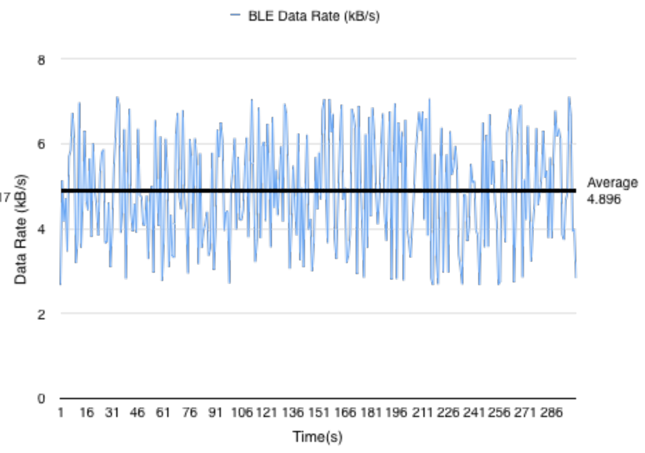


Figure 6.2: BLE Data Rate

Figure 6.1 and 6.2 shows the data rate of Wi-Fi Direct and BLE respectively. From the result, we can see that the bandwidth of Wi-Fi Direct was about 19MB/s and the data speed of BLE was only around 5kB/s. Moreover, from the figures we can see the data rate of Wi-Fi Direct was more stable than the data rate of BLE. Therefore, since BLE was only used for cross-platform communication and other scenarios when Wi-Fi

Direct can not be supported, this paper will only present Wi-Fi Direct for the remaining of the experiments.

Before continuing to the remaining experiments, there are some definitions of terminologies that need to be clarified:

1. *Throughput* - How much data can a node process and send to other nodes per second.
2. *Packages per Second* - How many packages (either requests, responses, or raw network packages) can a node process and send to other nodes per second.
3. *Hit Rate* - During the procedure of rebalancing and failover, some of the ITSs might be temporarily offline, therefore accessing tuples in those offline ITSs will result in failures. *Hit Rate* indicates how many operations can successfully reach the desired tuples during the rebalancing or failover procedure. For example, hit rate 0.95 means 95 percent of the operations can successfully access the desired tuples.

6.2 Basic Throughput

The second experiment tested the throughput between two nodes without the intervention of the coordination framework. This experiment will help with evaluating the performance of the coordination framework. When the coordination framework is working, if the throughput of a node is still relatively close to the throughput when the coordination framework is offline, it means the performance of the coordination model is high. This is because its key components like data routing (i.e. consistent hashing), CoAP messages packing/unpacking, resource locating, and request processing are not the bottleneck compared to the physical network capability.

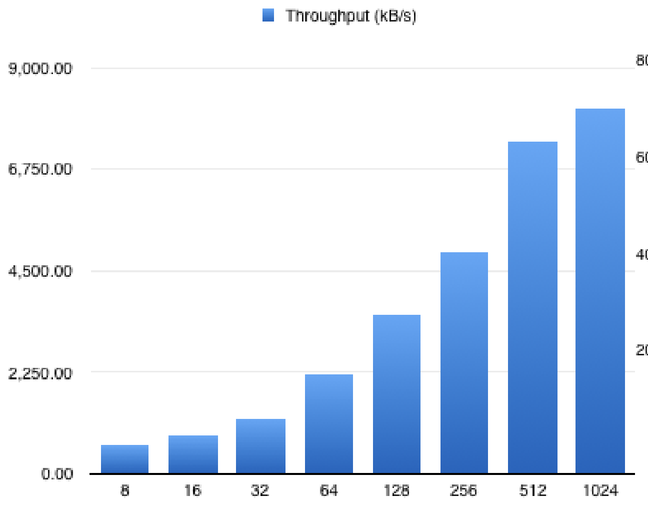


Figure 6.3: Throughput Between 2 nodes

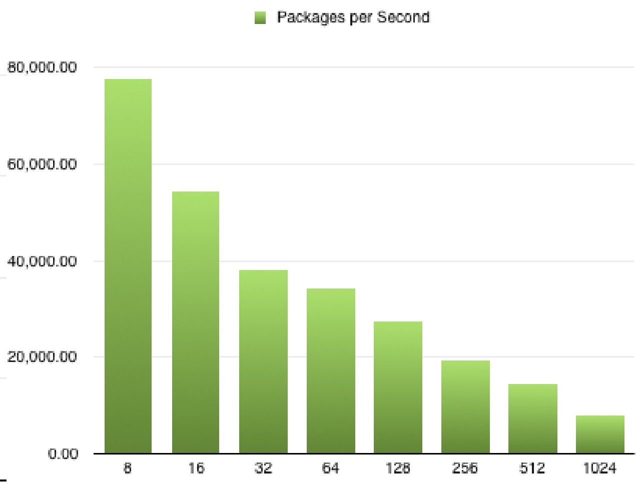


Figure 6.4: Packages per Second Between 2 nodes

Figure 6.3 shows the throughput of the network when carrying different sizes of payload. When dispatching a network package, the network streaming API will still need some time to prepare the stream, thus the throughput will be smaller than the previous data rate when the packages have been limited to a certain size. The framework is designed for facilitating the communication of IoT scenarios. Hence this study has restricted the payload size to the most common sizes of IoT communication. Figure 6.3 and 6.4 reveal that the smaller the package is, the more packages the network can transmit. However, when the package size is small, the throughput is worse than in the case of bigger package size.

6.3 Throughput of the Coordination Framework

I first tested the throughput of the coordination framework when there are only two nodes in the system. Each node contains 10 ITSs.

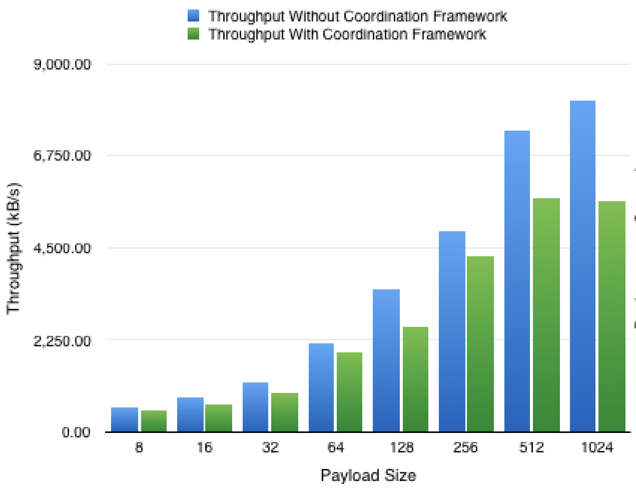


Figure 6.5: Throughput of the Coordination Framework (2 Nodes in the System)

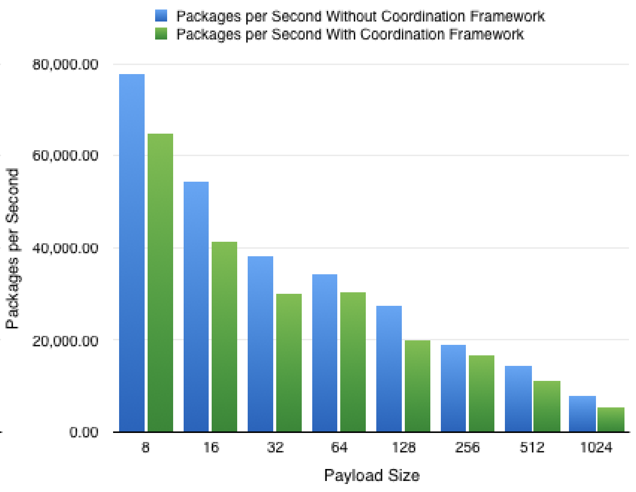


Figure 6.6: Packages per Second of the Coordination Framework (2 Nodes in the System)

The above figures show the performance of the coordination framework when there are only two nodes in the system. As revealed by figure 6.5 and 6.6, after the coordination framework is involved, the throughput was averagely impacted by 20%. Throughput was impacted in this way because the layers of the framework needed extra time to finish their jobs, such as calculating the destination, processing the requests, and packing/unpacking the CoAP messages.

The throughput of the coordination framework was then evaluated again with more devices. This time, the experiment system was formed by all the seven devices. Each device was holding 10 ITSs.

Figure 6.7 and 6.8 show the results of throughput and packages per second when there were 7 nodes in the system. The ratio between the throughput with and without the intervention of the framework is called the *Performance Ratio*. The closer the performance ratio is to 1, the higher performance the system has. Figure 6.9 highlights the different performance ratio of the system when it had 2 nodes and 7 nodes. From the figure we can see that the system had better performance when it contains more nodes. This is because that the coordination layer of the framework distributes requests to multiple nodes so that the requests can be processed in parallel. Therefore, adding nodes to the system will significantly multiply the power of it. The

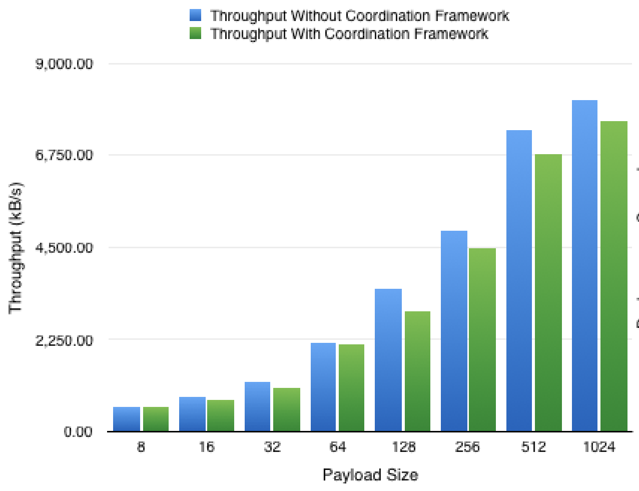


Figure 6.7: Throughput of the Coordination Framework (7 Nodes in the System)

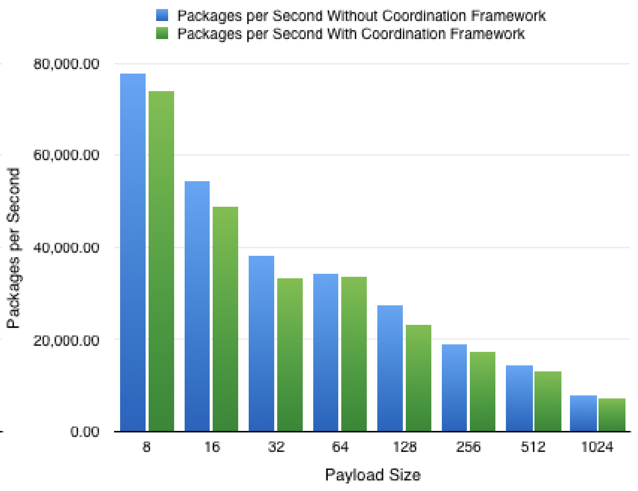


Figure 6.8: Packages per Second of the Coordination Framework (7 Nodes in the System)

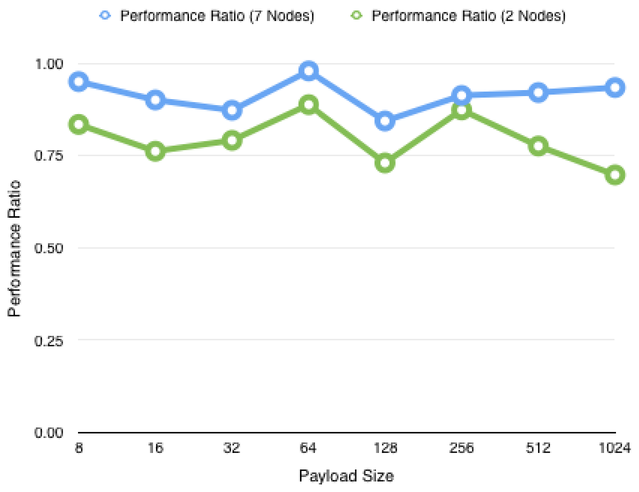


Figure 6.9: The Performance Ratio of the Coordination Framework

result proves that this coordination model and framework can combine multiple nodes together to become a more powerful system.

Moreover, from figure 6.9 we know that when there are 7 nodes in the system, the performance ratio is more than 0.9, which means the intervention of the coordination framework only slightly affects the throughput. Thus we can tell that the coordination framework is a high-performance framework.

6.4 Engagement and Disengagement

The engagement and disengagement have been evaluated from different aspects. During the tests, each time when a device was involved in an engagement or disengagement procedure, it recored how much time it costed to finish the procedure. After each test, the devices submitted the record to a server to aggregate and analyzed the data.

I first tested how much time the engagement and disengagement would cost when there was a different amount of preloaded data. The test started with 1000 tuples and ended with 1million tuples. Each tuple was limited to 32 bytes. Initially, the system contained 7 nodes. Then the system would randomly pick a node to disconnect and connect back again. Such a procedure was repeated 10,000 times to find out the most accurate time-cost.

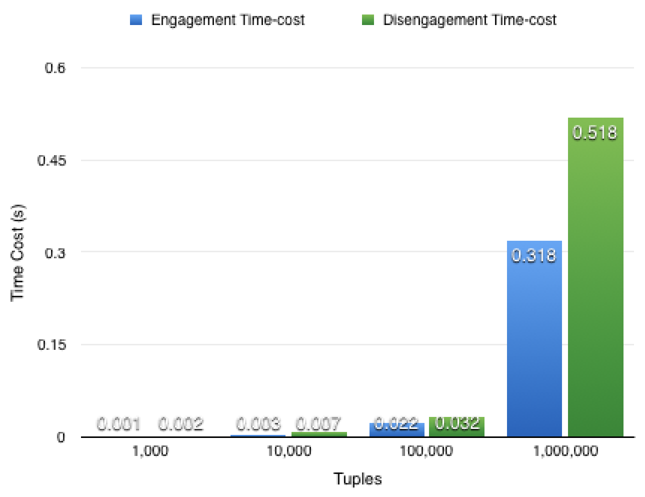


Figure 6.10: Average Time-cost for Engagement and Disengagement

Figure 6.10 shows that dealing with engagement is faster than dealing with disengagement. From the figure we can also discover that even if there are 1 million tuples in the federated tuple space, engagement and disengagement can still be finished in a relatively short time, which proves the efficiency of the engagement and disengagement design.

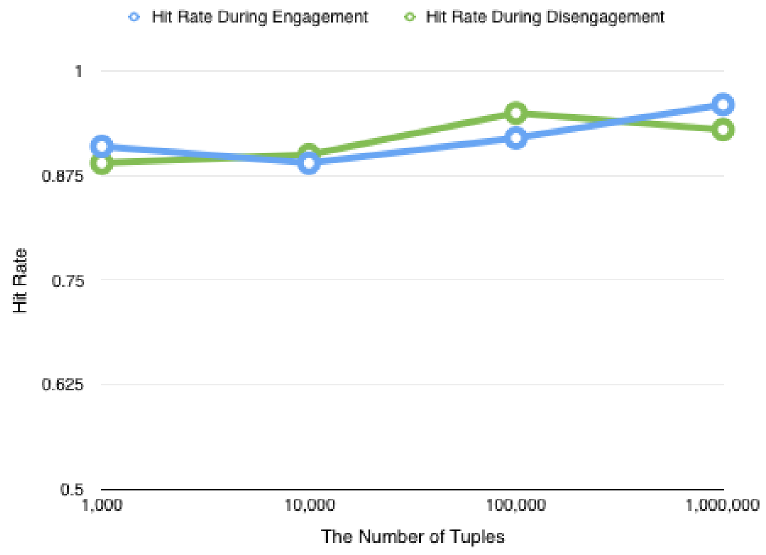


Figure 6.11: Hit Rate During Engagement and Disengagement

Figure 6.11 uncovers the hit-rate when dealing with engagement and disengagement. From the figure we can see that because of the rebalancing and failover procedure, some of the tuples became temporarily offline. However, the hit-rate can still be kept at a high level (over 90%), and recalling that the engagement and disengagement procedure can be finished within a short time, such impact is still acceptable.

6.5 Data Distribution

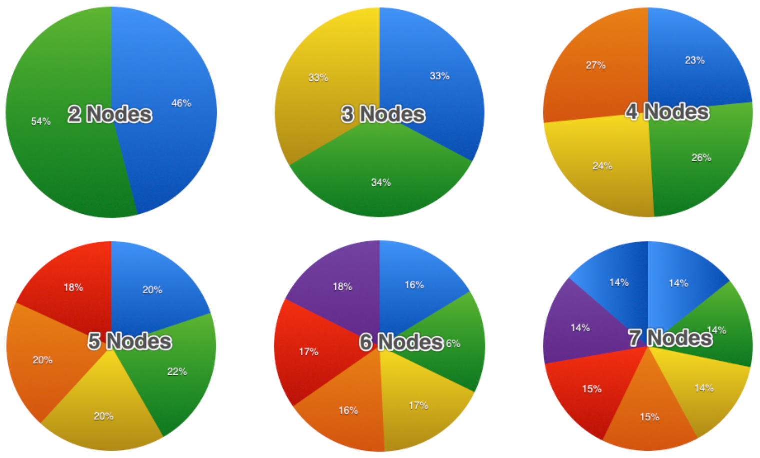


Figure 6.12: Data Distribution of the System

As mentioned above, the framework also needs to guarantee all the data will be evenly distributed to prevent a hot spot. A hot spot means it carries significantly more data than other nodes in the system and more requests will be routed to this node. Consequently, this node will become the bottleneck of the system. Therefore, this study observes the data distribution when there is a different number of nodes in the system. 1 million tuples with random names are loaded into the system.

Figure 6.12 shows that when there are two nodes involved in the systems, 54% of the tuples have been distributed to a node while another node holds the remaining 46%. When there are three nodes in the system, each node carries approximately 1/3 of the total tuples. The same result can be found when there are more nodes in the system. These results prove that data can be averagely distributed to the involving nodes and the more nodes the system has, the more linear the data distribution is.

Chapter 7
Summary and Future Work

This chapter will briefly summarize the contribution of this proposed research and illustrates what is yet to be accomplished.

7.1 Summary

The main contributions of this thesis are:

- Proposal of a high-performance and fault-tolerant coordination model.
- Based on the coordination model, it presents a cross-platform coordination framework that can significantly facilitate the development of distributed mobile applications.

Mobile devices have increasingly become more important in the daily lives of individuals and groups. As a result, initiatives on the development techniques for distributed mobile applications have significantly increased. The coordination model is one of the most important components of developing a distributed mobile application. A number of resources have been directed on this area already, with LIME being one of the most remarkable masterpieces to come from it all. This paper first introduces the background information on the history of the mobile application development. After that, this paper reviews the LIME coordination model and its predecessor the Linda coordination model. After evaluating the strengths and shortcomings of these coordination models, this research acknowledges that the performance issue is the most critical flaw that prevents them from being widely used. In addition, this research points out that fault-tolerance is another important requirement for developing modern distributed mobile applications. Aiming to overcome the shortages of LIME as well as provide fault-tolerance support, this paper proposes a coordination model that leverages the advantages of consistent hashing and designs a creative replica mechanism. Consistent hashing can help the system obtain a higher throughput and lower latency time while the self-defined replica mechanism can guarantee availability and data integrity of the system. Based on the proposed coordination model, a coordination framework using P2P technologies and CoAP protocol has been introduced to facilitate the development of distributed mobile applications. The result of several experiments can prove the goals set forth of designing the coordination model and framework have been successfully achieved.

This research paper is organized in the following structure:

- **Why do we need a new coordination model**

  In a distributed mobile application, a coordination model will be responsible for coordinating all the devices involved to work properly in unity for a single application. LIME (Linda In a Mobile Environment) is the most fully-fledged coordination model for the mobile environment. Based on its predecessor Linda, LIME introduces many appealing concepts such as transient sharing, location-aware computing, and reactive statements to accommodate itself into the mobile environment. After carefully reviewing and examining LIME, this paper pinpoints the reasons that drastically restrict the performance of LIME, which are:

  - The transient sharing and location-aware computing features of LIME require it to go over all the tuple spaces in the system during each engagement and disengagement. Such design can lead to an unbearable processing flood to the system when dealing with a large amount of engagements and disengagements.
  - The over-designed LIMESystem ITS will make the system pay a lot of efforts on the synchronization.
  - The pattern matching mechanism used by LIME can severely affect the performance for locating a tuple if the tuple is defined by complex patterns.

- **How to improve the performance of the coordination model**

  Focusing on developing a coordination model with better performance, this paper utilizes the feature of consistent hashing to eliminate the performance issue of LIME. The idea of consistent hashing can significantly reduce the time cost for locating the tuples, and simplify the procedure of engagement and disengagement. Instead of directly adopting consistent hashing to the new coordination model, this paper optimizes the consistent hashing in many ways so that it can provide the best support for performing the coordination. For example, hosts will no longer be mapped to the consistent hashing ring so that the coordination model can support both physical mobility and logical mobility. Moreover, this paper designs a lightweight global object called the SystemRuntime to eliminate the performance affect caused by the over-designed LIMESystem ITS.

- **How to make the coordination model fault-tolerant**

  Performance is not the only requirement for developing a coordination model. Fault-tolerance is another important issue that demands equal attention. Fault-tolerance involves two aspects: the availability and the data integrity. The availability means the system needs to keep functioning in the event that some of the nodes are encountering errors. The data integrity means the engagement and disengagement of nodes cannot result in any data lost. The coordination model guarantees the fault-tolerance feature by inventing a robust replica mechanism. This replica mechanism can automatically generate a backup of the whole system and dispatch the backup to all the participating nodes. The replica mechanism enables the system to be fault-tolerant by constantly monitoring the system and automatically launching the recovering process when node failures or errors occur.

- **How to facilitate the development of distributed mobile applications**

  To facilitate the development of mobile applications and minimize the mistakes made by programmers, this study implemented a cross-platform development framework based on the proposed coordination model. On the physical level, it utilized Wi-Fi Direct and Bluetooth Low Energy technologies to form the P2P network. On the logical level, it customized the CoAP protocol to optimize the processing performance and networking communications. It also adopted many technologies and algorithms to obtain better performance such as thread pool, round-robin scheduling, workflow management, hash table, and the binary search tree.

- **The Results**

  Several experiments show that the bottleneck of the proposed approach is not the coordination model but the network, which proves that the proposed coordination model is high-performance. Furthermore, statistic data about the hit-rate during engagement and disengagement procedures also proves that the coordination model is robust enough to guarantee the data integrity and system availability.

7.2 Future Work

The coordination model and framework is expected to be improved in the following aspects:

7.2.1 Engagement and Disengagement

Engagement

Experiments demonstrated that, during the engagement procedure, moving a large number of tuples between nodes might reduce the hit rate. This reduction occurs when an ITS begins balancing from a hosting node to another node, causing all the tuples carried by the ITS to be temporarily unavailable until the transmission of the ITS is finished. If the data set is relatively small, the hit rate might not be affected because the transmission of ITSs can be finished quickly. However, when there is a large amount of data storing in the ITS that is going to be transmitted, it is likely that some nodes might attempt to access the tuples stored in the ITS. On the other hand, because ITSs are migrated sequentially, balancing a large mount of data would take a relatively long time to finish. This extended amount of time could result in a drastically inscreased possibility of encountering a false hit. Moreover, when an ITSs is being transferred, operations on the ITS will result in failure because the ITS is temporarily unavailable.

This problem can be resolved in three steps.

First, instead of moving ITSs one by one from a node to another and synchronizing the SystemRuntime each time an ITS is migrated, the framework will copy the ITSs to the destination node and only synchronize the SystemRuntime one time after all the ITSs have been successfully replicated. The original copy will not be removed until the whole rebalancing procedure is finished. Therefore, all the related requests will still be routed to the original node during rebalancing.

Second, parallel rebalancing will be introduced to enable multiple ITSs to be transferred simultaneously. There will be several threads performing the rebalance at the same time so that the time for rebalancing will be significantly reduced.

Finally, in order to accept operations even when the target ITS is being rebalanced, the framework will introduce the operation log mechanism. Such mechanism will record every operation after the rebalance of the ITS is started. Once the rebalance of the ITS finishes, all the operations in the operation log will be reproduced in the new node.

Disengagement

When the system is performing failover for the leaving node, data storing in the leaving node can only gradually become available again because it takes time to load all the content from the replica, leaving only the loaded content available to access. This will impact the hit rate and block operations on the ITSs being reproduced.

This problem can be solved by allowing the replica to instantly become available after a node leaves the system and all the related operations will be redirected to the node that stores the replica of the leaving node until the failover procedure is finished. This operation log mechanism will also be applied to the disengagement procedure to guarantee the availability of the ITSs. Besides, similar to engagement, the failover procedure will be performed by several threads simultaneously to minimize the time-cost as well.

7.2.2 Replica Mechanism

There are some foreseeable problems of the current replica mechanism.

First, storing the replica demands extra storage space from the devices. When the data set becomes considerably large, constrained devices might not have enough space to store the replica. Second, when a node leaves the system, it is possible that the node storing its replica could disappear at the same time or before the failover is finished. A node disappearing at an inappropriate time will cause data loss and affect the data integrity.

The above problems can be solved by applying three mechanisms to the coordination model.

1. The replica will be compressed when its size reaches a predefined threshold. Several compression/decompression algorithms will be observed and evaluated to find the best trade-off between the compressing/decompressing speed and the compression ratio.

2. Each host will have multiple replicas so that the system can be more robust.

3. The framework will adopt a better strategy to manage the replicas. Only the most powerful nodes (e.g. smartphones and tablets) and the nodes that have the least possibility to leave the system will be assigned to store the replicas. Constrained devices (e.g. sensors and IoT terminals) and nodes that will frequently join and leave the system will not carry any replica.

7.2.3 SystemRuntime Synchronization

In the current implementation of the coordination framework, synchronizing the global SystemRuntime object will retrieve multiple copies of the SystemRuntime object from other nodes. Although the SystemRuntime object has been optimized to be small and retrieving it can be done in a relatively short time, this procedure can be accomplished at an even greater speed. Instead of retrieving the copies from other nodes, only the MD5 value of the SystemRuntime object will be transmitted to compare whether the remote copy is different than the local SystemRuntime object. Remote copy needs to be retrieved only when the MD5 values are not the same. Typically, the size of a SystemRuntime object is around 1kB, while an MD5 value costs only 16 bytes. Hence the performance of synchronizing the SystemRuntime object can be considerably improved.

7.2.4 Support More Language

The coordination framework is currently fully implemented in C#. It was partially implemented in C during the aforementioned IoT project. Due to the fact that most of the constrained devices are based on Linux where C# can hardly be supported, other language versions (e.g. python, C, java) of the coordination framework will be implemented in the future.

References

[1] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli, "Mobile-agent coordination models for internet applications," *Computer*, vol. 33, no. 2, pp. 82–89, 2000.

[2] Paolo Ciancarini, "Coordination models and languages as software integrators," *ACM Computing Surveys (CSUR)*, vol. 28, no. 2, pp. 300–302, 1996.

[3] George A Papadopoulos and Farhad Arbab, "Coordination models and languages," *Advances in computers*, vol. 46, pp. 329–400, 1998.

[4] Google Trends, "Search interests on mobile cloud computing," sept 2015. [Online]. Available: https://www.google.com/trends/explore?date=all&q=mobile%20cloud%20computing

[5] IDG Global Solutions, "Idg global mobile 2014 survey," *London: IDG Global Solutions*, 2014.

[6] Sophos Lab, "Infographic: Users weighed down by multiple gadgets – survey reveals the most carried devices," mar 2013. [Online]. Available: https://nakedsecurity.sophos.com/2013/03/14/devices-wozniak-infographic/

[7] KPCB, "2016 internet trends report," jun 2016. [Online]. Available: http://www.kpcb.com/blog/2016-internet-trends-report

[8] Morgan Stanley, "The mobile internet report," *Morgan Stanley research*, 2009.

[9] Adam Lella, "Number of mobile-only internet users now exceeds desktop-only in the u.s." April 2015. [Online]. Available: https://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S

[10] Ted Lewis, "Information appliances: Gadget netopia," *Computer*, vol. 31, no. 1, pp. 59–68, 1998.

[11] Ekaterina Chtcherbina and Marquart Franz, "Peer-to-peer coordination framework (p2pc): Enabler of mobile ad-hoc networking for medicine, business, and entertainment," in *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet, L'Aquila, Italy*, 2003, pp. 883–891.

[12] Dejan S Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu, "Peer-to-peer computing," 2002. [Online]. Available: http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf

[13] David Gelernter and Nicholas Carriero, "Coordination languages and their significance," *Communications of the ACM*, vol. 35, no. 2, p. 96, 1992.

[14] Amy L Murphy, Gian Pietro Picco, and G-C Roman, "Lime: A middleware for physical and logical mobility," in *Distributed Computing Systems, 2001. 21st International Conference on.* IEEE, 2001, pp. 524–533.

[15] Sudhir Ahuja, N Curriero, and David Gelernter, "Linda and friends," *Computer;(United States)*, vol. 19, no. 8, 1986.

[16] Gian Pietro Picco, Amy L Murphy, and Gruia-Catalin Roman, "Lime: Linda meets mobility," in *Proceedings of the 21st international conference on Software engineering.* ACM, 1999, pp. 368–377.

[17] Thomas W Malone and Kevin Crowston, "The interdisciplinary study of coordination," *ACM Computing Surveys (CSUR)*, vol. 26, no. 1, pp. 87–119, 1994.

[18] Gruia-Catalin Roman, Amy L Murphy, and Gian Pietro Picco, "Coordination and mobility," in *Coordination of Internet agents.* Springer, 2001, pp. 253–273.

[19] Amy L Murphy, Gian Pietro Picco, and Gruia-Catalin Roman, "Lime: A coordination model and middleware supporting mobility of hosts and agents," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 3, pp. 279–328, 2006.

[20] Rüdiger Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on.* IEEE, 2001, pp. 101–102.

[21] HMN Dilum Bandara and Anura P Jayasumana, "Collaborative applications over peer-to-peer systems–challenges and solutions," *Peer-to-Peer Networking and Applications*, vol. 6, no. 3, pp. 257–276, 2013.

[22] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[23] David Barkai, *Peer-to-Peer Computing: technologies for sharing and collaborating on the net.* Intel Press, 2001.

[24] Ralf Steinmetz and Klaus Wehrle, "2. what is this "peer-to-peer" about?" in *Peer-to-Peer Systems and Applications.* Springer, 2005, pp. 9–16.

[25] Christopher Lueg and Danyel Fisher, *From Usenet to CoWebs: interacting with social information spaces.* Springer Science & Business Media, 2012.

[26] Michael Hauben, "The social forces behind the development of usenet," *Netizens: An Anthology*, 1996.

[27] Geoffrey Fox, “Peer-to-peer networks,” *Computing in Science & Engineering*, vol. 3, no. 3, pp. 75–77, 2001.

[28] Parul Sharma, Anuja Bhakuni, and Rishabh Kaushal, “Performance analysis of bittorrent protocol,” in *Communications (NCC), 2013 National Conference on.* IEEE, 2013, pp. 1–5.

[29] Mina Kamel, Caterina Scoglio, and Todd Easton, “Optimal topology design for overlay networks,” in *International Conference on Research in Networking.* Springer, 2007, pp. 714–725.

[30] Imen Filali, Francesco Bongiovanni, Fabrice Huet, and Françoise Baude, “A survey of structured p2p systems for rdf data storage and retrieval,” in *Transactions on large-scale data-and knowledge-centered systems iii.* Springer, 2011, pp. 20–55.

[31] Ann Chervenak and Shishir Bharathi, “Peer-to-peer approaches to grid resource discovery,” in *Making Grids Work.* Springer, 2008, pp. 59–76.

[32] Xing Jin and S-H Gary Chan, “Unstructured peer-to-peer network architectures,” in *Handbook of Peer-to-Peer Networking.* Springer, 2010, pp. 117–142.

[33] Xuemin Sherman Shen, Heather Yu, John Buford, and Mursalin Akon, *Handbook of peer-to-peer networking.* Springer Science & Business Media, 2010, vol. 34.

[34] Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya, “Peer-to-peer-based resource discovery in global grids: a tutorial,” *IEEE Communications Surveys & Tutorials*, vol. 10, no. 2, pp. 6–33, 2008.

[35] Deng Li, Hui Liu, and Athanasios Vasilakos, “An efficient, scalable and robust p2p overlay for autonomic communication,” in *Autonomic Communication.* Springer, 2009, pp. 327–350.

[36] HMN Dilum Bandara and Anura P Jayasumana, “Evaluation of p2p resource discovery architectures using real-life multi-attribute resource and query characteristics,” in *2012 IEEE Consumer Communications and Networking Conference (CCNC).* IEEE, 2012, pp. 634–639.

[37] Kevin Curran, Amanda Millar, and Conor Mc Garvey, “Near field communication,” *International Journal of Electrical and Computer Engineering*, vol. 2, no. 3, p. 371, 2012.

[38] Brian P Crow, Indra Widjaja, LG Kim, and Prescott T Sakai, “Ieee 802.11 wireless local area networks,” *IEEE Communications magazine*, vol. 35, no. 9, pp. 116–126, 1997.

[39] WiFi Alliance, “Who we are - history of wifi alliance,” jan 1999. [Online]. Available: http://www.wi-fi.org/who-we-are/history

[40] WiFi Alliance, “Ieee802.11y-2008,” sep 2008. [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.11y-2008

[41] WiFi Alliance, “Wifi.” [Online]. Available: https://en.wikipedia.org/wiki/Wi-Fi

[42] Daniel L Lough, David J Robinson, and Ian G Schneller, “Ieee 802.11 security,” *Network Security: Current Status and Future Directions*, pp. 297–312, 2007.

[43] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano, “Device-to-device communications with wi-fi direct: overview and experimentation,” *IEEE wireless communications*, vol. 20, no. 3, pp. 96–104, 2013.

[44] Wi-Fi Alliance, “Wi-fi certified wi-fi direct: Personal, portable wi-fi,” *Wi-Fi Alliance*, 2014.

[45] Arash Asadi and Vincenzo Mancuso, “Wifi direct and lte d2d in action,” in *Wireless Days (WD), 2013 IFIP*. IEEE, 2013, pp. 1–8.

[46] Wi-Fi Alliance, “Wi-fi certified wi-fi direct,” *White paper*, 2010.

[47] ABI Research, “Global wi-fi direct device shipments,” April 2014. [Online]. Available: https://www.abiresearch.com/

[48] Michael Stauffer, “Connectivity solutions for smart tvs,” in *Consumer Electronics-Berlin (ICCE-Berlin), 2012 IEEE International Conference on.* IEEE, 2012, pp. 245–249.

[49] Phillip A McCoog and Steve Claiborne, “Communication architectures for direct printing and scanning,” 2015, uS Patent 9,179,016.

[50] Jott Messenger, “About jott messenger.” [Online]. Available: https://jott.com/

[51] Spaceteam, “About spaceteam card game.” [Online]. Available: http://www.playspaceteam.com/

[52] SIG Bluetooth, “Bluetooth specification version 1.1,” *Available HTTP: http://www. bluetooth. com*, 2001.

[53] James Chen, “Fast connection establishment method for bluetooth device,” Dec. 20 2005, uS Patent 6,978,119.

[54] SIG Bluetooth, “Bluetooth specification version 4.2,” *Bluetooth SIG*, 2014.

[55] Bluetooth SIG, “Bluetooth® 5 quadruples range, doubles speed, increases data broadcasting capacity by 800https://www.bluetooth.com/news/pressreleases/2016/06/16/-bluetooth5-quadruples-rangedoubles-speedincreases-data-broadcasting-capacity-by-800

[56] Bluetooth SIG, “Our history - the history of the bluetooth special interest group,” sep 2016. [Online]. Available: https://www.bluetooth.com/about-us/our-history

[57] SIG Bluetooth, “Bluetooth core specification version 4.0,” *Specification of the Bluetooth System*, 2010.

[58] Bluetooth Specification Version, “1.0 b,” *Service Discovery Protocol*, pp. 324–384, 1999.

[59] SIG Bluetooth, “Bluetooth specification version 2.0+ edr,” *Bluetooth SIG Standard*, 2004.

[60] SIG Bluetooth, “Bluetooth core specification version 2.1+ edr,” *Specification of the Bluetooth system*, 2007.

[61] SIG Bluetooth, “Bluetooth specification version 3.0,” *Bluetooth Special Interest Group Std*, 2009.

[62] Leonard Richardson and Sam Ruby, *RESTful web services.* ” O’Reilly Media, Inc.”, 2008.

[63] Xinyang Feng, Jianjing Shen, and Ying Fan, “Rest: An alternative to rpc for web services architecture,” in *Future Information Networks, 2009. ICFIN 2009. First International Conference on.* IEEE, 2009, pp. 7–10.

[64] Roy Fielding, “Fielding dissertation:architectural styles and the design of network-based software architectures - chapter 5: Representational state transfer (rest),” *Recuperado el*, vol. 8, 2000.

[65] Carsten Bormann, Angelo P Castellani, and Zach Shelby, “Coap: An application protocol for billions of tiny internet nodes,” *IEEE Internet Computing*, vol. 16, no. 2, p. 62, 2012.

[66] Zack Shelby, Klaus Hartke, and Carsten Bormann, “The constrained application protocol (coap)(rfc 7252),” 2014.

[67] Wojciech Galuba and Sarunas Girdzijauskas, “Distributed hash table,” in *Encyclopedia of Database Systems.* Springer, 2009, pp. 903–904.

[68] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin, “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web,” in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.* ACM, 1997, pp. 654–663.