Enabling Traceability in Large-Scale RFID Networks

A dissertation submitted in fulfillment

of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

Yanbo Wu

Supervisor: Prof. Hong Shen, Dr. Quanzheng Sheng

School of Computer Science

The University of Adelaide

December, 2011

Table of Contents

List of Figures

List of Tables

Abstract of the Dissertation

Enabling Traceability in Large-Scale RFID Networks

by

Yanbo Wu

Doctor of Philosophy in Computer Science

The University of Adelaide, 2011

The emergence of radio frequency identification (RFID) technology brings significant social and economic benefits. As a non line of sight technology, RFID provides an effective way to record movements of objects within a networked system formed by a set of distributed and collaborating parties. A trail of such recorded movements is the foundation for enabling traceability applications. While traceability is a critical aspect of the majority of RFID applications, realizing traceability for these applications brings many fundamental research and development issues, including storage efficiency, query processing complexity, privacy etc.

In this dissertation, we present a novel approach to realize RFIID-based traceability in large, autonomous and heterogeneous distributed networks. We first propose a Peer-to-Peer (P2P) architecture, namely PeerTrack. PeerTrack does not require any kind of centralized database for the RFID data or their index, neither it requires RFID data to be fully shared to partners. In PeerTrack, only a specific portion of data is requested by partners, when the access is necessary.

We introduce a distributed model, namely MOODS (a **M**odel for m**O**ving **O**bjects in **D**iscrete **S**pace), for the essential data structures of traceability.

MOODS is maintained by a distributed index on the top of a structured Peer-to-Peer overlay. We then propose efficient algorithms for the maintenance of MOODS. The algorithms are optimized to consume statistically minimal cost of bandwidth. Based on this model, we propose algorithms for efficient item-level and statistical traceability query processing.

We also propose a traceability mining model for distributed RFID streams, namely TISH (**T**ilted T**I**me **F**rame of **H**istogram). TISH takes advantages of two important data mining tools, namely *Tilted Time Series* and *Histogram*, and combines them to describe the patterns of RFID streams in the dimensions of both time and space, and capture the dynamicity of the patterns. We propose efficient algorithms to maintain TISH and algorithms that use it for traceability query processing and RFID stream mining.

We present a platform, namely PeerTrack Cloud, to bring the aforementioned RFID data modeling and traceability query processing techniques to the Cloud Environments. The platform features specific traceability-oriented modules for real-time query processing and efficient data storage.

The techniques proposed in this dissertation are implemented in "Asset Management System", which is a collaborative project with a local company. Finally, we conduct extensive performance studies of the proposed techniques. The experimental results reveal that our system i) is more scalable and outperforms the centralized approach when the data volume or the network becomes larger; ii) provides powerful programming interfaces for query processing; iii) is economy in both storage and bandwidth; and iv) can be easily adopted in cloud computing platforms.

Originality Statement

"Yanbo Wu certifies that this work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the Universitys digital research repository, the Library catalogue, also through web search engines, unless permission has been granted by the University to restrict access for a period of time."

Yanbo Wu

December 1st, 2011

To my mother and father,

who made all of this possible,

for their endless encouragement and support.

Acknowledgments

It has been a great pleasure working with the faculty, staff, students at the University of Adelaide, during my tenure as a doctoral student, and I would like to thank them all for such a great graduate school experience. My foremost thank goes to my thesis supervisor Dr. Quanzheng Sheng, a talented teacher and passionate scientist. Dr. Sheng instilled a thirst for excellence in me, taught me how to do high-quality research, and helped me think independently and creatively. He not only guided my research, but also served as a mentor and role model as I embarked on my academic career. I will forever cherish his full support during my study. I also would like to thank Prof. Hong Shen, who gave me many valuable suggestions to my research.

I thank my co-authors: Quanzheng Sheng, Hong Shen, Sherali Zeadally, Jian Yu, Damith Ranasinghe and Jun Han, for their productive and enjoyable collaborations. I would like also to thank anonymous reviewers for their valuable comments on earlier drafts of my papers.

I would like to thank my mother and father for their constant support and encouragement. They have been always being there whenever I needed them. I also would like to thank my dearest friend, Chenke Yang, for not only giving me valuable suggestions to my papers, but also supporting me spiritually.

Finally, I express my sincere appreciation to the University of Adelaide, who provided the Adelaide University Fee Scholarship (AFSI) and ARC-grant Funded Scholarship and to Google Inc., who provided the Google PhD Top-up Scholarship, to financially support my work in this dissertation.

Chapter 1

Introduction

Traceability refers to the capability to track the states (e.g. location) of goods, discover information regarding its past state and potentially estimate the future state. In the era of globalized economic, traceability becomes vital for efficient business operations and decision making. It is fundamental to a wide range of business applications such as inventory control, distribution planning, product recalls and counterfeit detection.

Identification technology is the bridge to connect the physical world with the digital one. Accurate and efficient identification is very important to realize traceability applications. RFID (Radio Frequency Identification) is capable of automatically extracting information from microelectronic tags attached to objects using radio waves. The identification is wireless and does not require the line of light. RFID was first explored in 1940s [Lan05]. However, due to its high cost and immaturity, its usage was limited in applications of small scale, such as automatic checkouts and electronic toll collection.

In the past decade, research initiatives by academic organizations such as the Auto-ID Labs[1], industrial interests from companies (e.g., Wal-Mart) and government initiatives (e.g., the United States Department of Defense) have rapidly es-

[1] http://www.autoidlabs.org

calated new developments and interests in RFID technology. Alongside, Moore's law has ensured that integrated circuits reduce in size, cost and power consumption. Consequently, RFID systems have become more reliable, efficient and more importantly, have become cheaper. These developments have resulted in an explosion in the number of RFID systems and applications.

"Networked RFID" [SLZ08b] is one of the important technological advances that make the explosion possible. The basic idea behind "Networked RFID" is to connect otherwise isolated RFID systems and other software. RFID tags only carry an unambiguous ID, meanwhile other data pertaining to the objects, including the past and current states are stored and accessed over the Internet. "Networked RFID" brings significant and promising benefits to traceability applications. For example, it makes it possible for applications to automatically analyze recorded RFID events to discover the current or past information of an object, without physical access to it. Many organizations are planning or already exploiting RFID to enable traceability. Wal-Mart, the world's largest public corporation by revenue, in 2005, mandated its top 100 suppliers to tag their pallets and cases using RFID [Ang05]. The U.S. Department of Defense released a policy on the use of RFID to its external suppliers and for internal operations in July of 2005 [RCT06].

However, to reap such benefits, researchers must overcome a number of key challenges. RFID-enabled traceability is not a single-layer problem. First of all, large-scale global RFID networks have the potential to generate unprecedented amounts of data. An important challenge therefore centers on the efficient management and sharing of the data within traceability applications. The system architecture must be scalable in order to deal with the data collected from networked RFID systems. For efficient processing and storage, data models must be

carefully considered. To allow business users to make decisions and analysis, we must support various kinds of queries for tracking and tracing each individual object, and mining the RFID steams. Finally, RFID is a pervasive technology that can unobtrusively monitor the movement of tagged goods or persons to generate sensitive data. As a result, privacy and security concerns must be addressed to allow wide-scale real world adoption.

This chapter is organized as follows. In Section 1.1, we briefly discuss RFID-enabled traceability in distributed environments. In Section 1.2, we present a typical traceability application and introduce the requirements for efficient traceability. In Section 1.3, we outline the research issues tackled in this dissertation. In Section 1.4, we summarize our contributions, and in Section 1.5, we describe the structure of this dissertation.

1.1 RFID Enabled Traceability

Traceability is an important requirement in many modern enterprises. For example, in food supply industry, the safety of food relies on the physical traceability throughout the chain. Finding the source of food deficiency is the key to stop its spread and avoid damages [KPD07]. In postal services, the ability to track individual packages is crucial to the success of delivery, and recovery when it is missing. In retailing business, traceability empowers the business managers to monitor the stock so that they can make rational decisions about supplies.

The existing solution for enabling traceability mostly use barcode as the identification method. This has several drawbacks. Firstly, line of light is required for a successful reading. An operator has to point the reader to the barcode label manually. This increases labor costs. Secondly, barcode cannot identify

the products of the same kind individually. Contrarily, it can only identify the type/category of the products. Thirdly, the amount of information which can be carried by barcode is very limited.

In the recent years, RFID is emerging as a promising technology for effective and efficient traceability system design. This is mainly motivated by three factors. Firstly, RFID tags are becoming cheaper and cheaper [Cho11]. It has become possible to affix RFID tags to goods at item level. Secondly, RFID is contact-less and it does not require line of light, so the identification process can be automatic. This significantly reduces the labor cost and identification time. Finally, RFID tags can carry more data than the competing technologies such as the barcode and the magnetic card. This makes it possible to record more information such as environmental data (temperature, humidity) for further analysis.

From a business perspective, RFID-enabled traceability can provide a number of benefits:

- It minimizes the human interaction in business processes. This does not only reduce the labor cost, but also reduce the possibility of human mistakes. More importantly, enabling traceability can help to pinpoint human mistakes.
- Without the delay caused by human contact, tracking and tracing can be done in real time. This makes it possible for fast reactions and better visibility of the business processes.
- Item-level traceability, enabled by RFID, is the foundation of many services, such as personalized service and accurate product recall. On the other hand, mining RFID streams can discover business patterns in a timely manner.

RFID-enabled traceability is a complicated problem which involves techniques

from multiple layers. It has been subject of much research in the past few years [CKR04a, SLZ08b, WRS11]. Adequate solutions to this problem will be very important for effective, efficient and intelligent business applications.

1.2 Supply Chain Management System: an Example

While the outcomes of our research are generic enough to be applicable to a wide range of applications, in this section, we use the *Supply Chain Management System* as an example to explain the problems and solutions in RFID-enabled traceability.

The motivation is based on the observations of the existing supply chain management system. According to the Supply Chain Management Review White Paper[2], the current Supply Chain Management (SCM) systems have a set of issues in transportation and delivery, including changeable rates caused by undetermined routes, inefficient shipping routes, lengthy timeline and security issues.

One of the reasons why these problems still exist after about half-century development of supply chains, is that the allocation of resources (transportation, human labor etc.) is not efficient. The deeper cause is that the link between these resources and their digital information is weak. Traceability, together with other analysis in SCM is designed to dig information out of an existing data warehouse [CE11]. In this way, the analysis is always behind the reality.

Another problem of existing SCM systems is that the personalized service is not good enough. Nowadays, more and more purchasing is done online. The customers wish to track where the products they bought are from and when they will be delivered. However, the tracking service is not satisfactory.

[2]http://www.scmr.com/article/current_trends_and_the_potential_for_automation_in_international_transporta

RFID technologies make the transformation of object flow to information flow easier. It has the potential to enable efficient, effective real-time traceability. Figure 1.1 shows the usage scenarios of a typical supply chain management system. All objects in the system is uniquely identified by RFID tags. RFID readers are placed at observation points, such as entrances and exits of warehouses. This system requires most of the traceability functionalities, including:

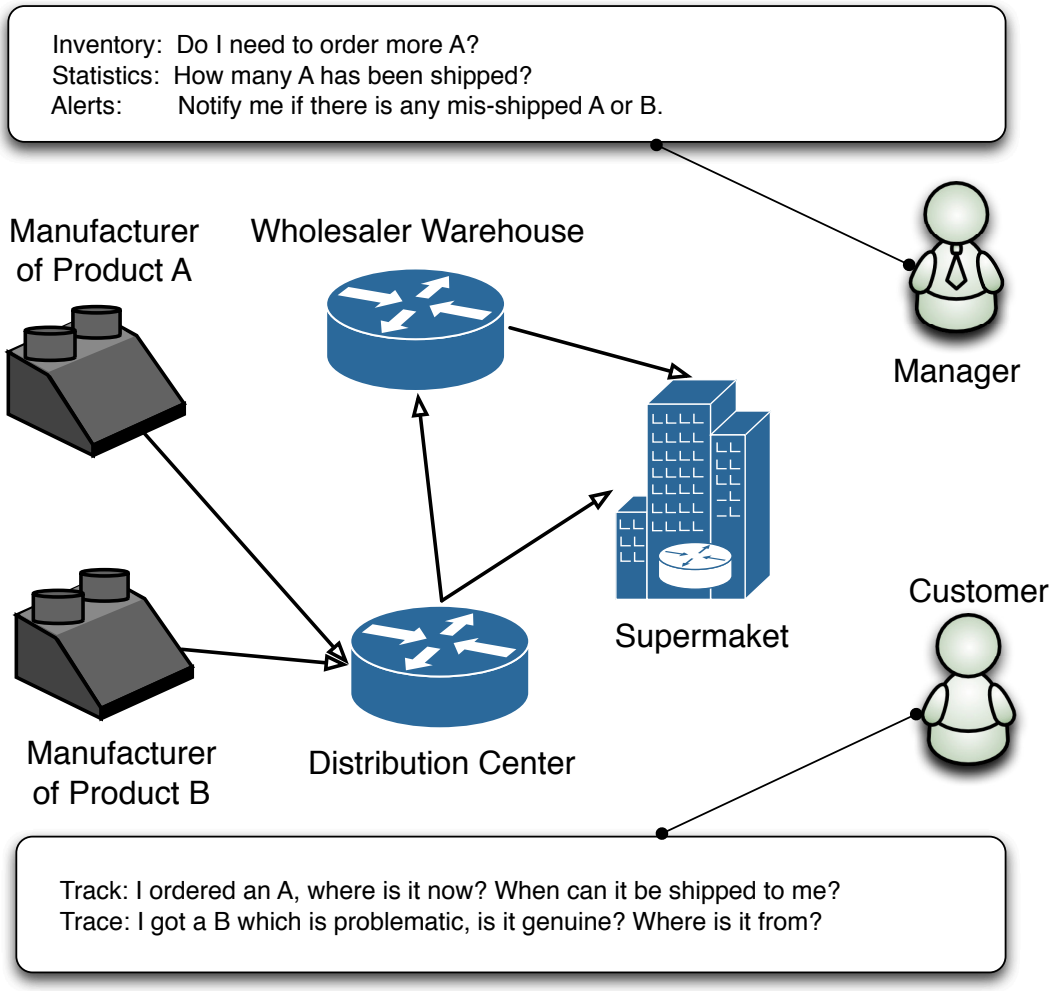


Figure 1.1: Supply Chain Management Scenarios

Real-time Tracking. The users of the system (including customers, managers or carriers) are able to find out where a particular item is by its id. For example, after placing an order online, a customer wants to get notified when the goods is shipped. He can also check the latest location of the shipped package whenever he wants. On the other hand, the retailer can get notified when something goes wrong, e.g., a misplaced delivery or missing package.

Tracing. A customer finds out that the product he received does not look genuine. He wants to find out where this product comes from so that he can ask the wholesaler or manufacturer for refund. Furthermore, the manufacturer is supposed to find out where all the products of the same kind are so he can call all of them back[3].

Business Intelligence. The managers of the retailer, or the wholesaler can make certain decisions about inventory or shipment arrangement, according to the real-time statistics of product sales, which can be monitored by the shipment of products to end users. Moreover, various mining or further analysis can be easily applied on the existing data models so that the managers can get a better understanding about the business process.

1.3 Research Issues

RFID-enabled traceability is a multiple layer problem. It is posed the following key challenges:

- **Adaptation to large and dynamic environments:** In a large traceable network, the membership of partners is continuously changing. For exam-

[3]This action requires the ability of both tracking and tracing.

ple, in a large supply chain network, new partners may join and existing partners may quit. Consequently, approaches that rely on static models are inappropriate. Instead, the system should be built on the top of a generic model which is adaptive to any network. Moreover, the model should be self-adaptive to the changes of the network.

- **Privacy and Sovereignty:** Large traceable networks often consist of partners from different organizations. They are reluctant to fully share their private data to others. However, without sharing, inter-organization traceability is impossible to achieve. Centralized architectures are not suitable because they require all the data to be uploaded to a centralized database and governed by a central administrator. Contrarily, in a Peer-to-Peer execution model, each partner physically owns his data. He can also control which part of the data to share, and share to whom. Peer-to-Peer computing is gaining a considerable momentum, as it naturally exploits the distributed nature of the Internet [YG01].

- **Efficient mining over distributed RFID streams:** In large networks, the records collected from RFID readers form a *stream* of large volume. Traditional data mining techniques which require to read the data for multiple times are not suitable. Also, in distributed environments, it is costly to access remote data during the mining process. As a result, the mining techniques used in RFID-enabled traceable networks should make as few remote accesses as possible.

- **Efficient traceability query processing:** Traceability queries (tracking, tracing, aggregation etc.) are the subset of location-based spatial queries [ZZP03]. However, existing query processing techniques cannot be used directly because there are significant differences between RFID and

normal spatial data. It is discrete, meanwhile spatial data is continuous. Moreover, Spatial data is more concerned about the geographical change (i.e., change of coordinations) of objects, meanwhile the location changes of RFID-tagged objects often imply business transactions.

1.4 Contribution Overview

We propose a generic Peer-to-Peer data model and architecture for large-scale RFID traceable networks. We propose traceability query processing algorithms and traceability mining algorithms. We also propose a cloud-based framework to make traceability as a service using our models. The proposed techniques are implemented in PeerTrack Platform and deployed in an application called *PeerTrack Asset Management System*. In particular, the main research contributions in this thesis focus on the following:

1.4.1 Peer-to-Peer Traceability Model and Architecture

We propose a generic data model to abstract and manage the basic and advanced data structures in RFID-enabled traceability. We also propose an architecture for large-scale traceable networks [SWR10, WSR11b]. This architecture is built on the top of a P2P overlay network so that no centralized database/index is required. It is purely distributed so that it is scalable in terms of the size of network and the volume of the RFID data.

We propose an algorithm to index the objects in the P2P network. Objects are indexed at deterministic gateway nodes that are responsible for updating objects' status at the source and destination nodes for their movements. In this way, the distributed traceability model is established and maintained.

To reduce the indexing overhead from massive volumes of data in large-scale applications, we further propose an enhanced group-based indexing approach. This approach takes both indexing efficiency and load balancing into consideration. We design a novel grouping algorithm which abstracts the key factor for indexing efficiency and load balancing as a single parameter. It can be used to determine the tradeoff between them. We also find its optimistic value and prove its validity.

For better load balancing in the situation that the system setting is static, we design a distributed data structure called *Prefix Triangle* which is a simplified version of *Distributed Hash Tree*. It uses three nodes instead one so that the indices are more evenly distributed in the network.

One of the important issues in Peer-to-Peer network is the handling of node failures. We design a replication-based failure handling approach. There are several different cases of replications, i.e., indices replication, metadata replication and path-data replication.

Finally, we propose algorithms for traceability query processing in the aforementioned architecture. The queries supported include item-level tracing and tracking queries and statistical queries.

1.4.2 Traceability Mining over Distributed RFID Streams

We propose a distributed traceability mining model to achieve scalable data mining in large-scale networks. Our model is a combination of two important tools in data mining, namely *Tilted Time Frame* and *Histogram*. Essentially, this model is the synopsis of the object flows. It represents the patterns of object flows among nodes for a long history using limited memory. The model suggests the probability that an object comes from a node at specific time.

We also develop algorithms to establish and maintain the model in a pure P2P fashion. To avoid long delays caused by network queries, we further develop an algorithm to choose the most possible neighbors as the targets of query rewriting.

This model can be applied to a wide range of applications. For data mining applications such as inventory/supply management system, it can be used as the base for further analysis such as time-based association rule learning and classification. It can also be used for heuristic traceability query processing. For example, query rewriting policy can be based on the distribution of objects from/to source/destination nodes. In this dissertation, we demonstrate its usage by introducing the TISH-based item-level tracing query processing algorithm.

1.4.3 Traceability as a Service

The "data-on-network" paradigm of RFID-based traceability applications has several shortcomings. One of them is that the RFID records are stored in several physical locations, especially in the P2P architecture. These locations may be distributed around the globe, thus it is likely that the query processing is with long latency. This can cause serious problems wherever a process requires fast system response.

To overcome this problem and reap the benefits of cloud computing, we design a cloud-based solution to make the traceability available as services to the general public, as well as enable fast query response. With this solution, the data is mirrored at the data center(s) which is (are) close to the nodes involved in the "chain of values". In this way, the data is available to the querier via a shorter network path than existing architectures. We develop algorithms of data partitioning and replication for various location-oriented data structures.

1.4.4 Implementation and Performance Study

We provide an implementation of proposed techniques inside the PeerTrack Platform. We adopt a number of state-of-the-art technologies for the implementation. We develop a service-oriented platform for traceability query processing. This work has been adopted in a real application for tracking and tracing returnable assets.

To validate the feasibility and benefits of our approach, we conduct extensive performance studies. The studies have been conducted from various aspects including scalability, network costs, load balancing and query processing efficiency.

1.5 Dissertation Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we introduce the background knowledge about RFID and traceability. And we present the state of the art in research about RFID-based traceability. We summarize the traceability requirements and traceability queries, and compare the existing approaches with respect to them.

In Chapter 3, we present the Peer-to-Peer traceability model and architecture. We first abstract the basic data structures and propose a distributed representation of them. We then propose the algorithms to maintain the data structures in the P2P fashion. Finally, we describe the algorithms of traceability query processing.

In Chapter 4, we give a detailed description of the traceability mining model. Firstly, we discuss the structure of the model and its benefits. Secondly, we propose the algorithms to maintain the model efficiently. In particular, we describe the usage of this model in traceability query processing in details.

In Chapter 5, we propose a design of cloud-based traceability network architecture based on the models we introduced in Chapter 3 and 4. In particular, we introduce the algorithm to efficiently replicate the RFID data for fast query response.

In Chapter 6, we describe the implementation of our approach for RFID-based traceability. We also report the results of a set of performance studies of our approach. Finally, in Chapter 7, we provide concluding remarks of this dissertation and discuss directions of future research.

Chapter 2

Background

In this chapter, we give an introduction to the research fields of traceability in RFID networks. We summarize the important traceability queries and the requirements of traceable RFID networks. We overview some major data models, data processing techniques, system architectures and standards for RFID traceable networks, to help readers gain a better understanding of the work described in this dissertation.

This chapter is organized as follows. In Section 2.1, we first introduce some basic concepts and technologies related to *Networked RFID*. In Section 2.2, we formally define *Traceability in Networked RFID* and identify a set of requirements. We also summarize a set of important traceability queries. In Section 2.3 and Section 2.4, we overview representative research on RFID data modeling and system architectures, and compare them with respect to our proposed requirements and queries. Finally, we summarize this chapter in Section 2.5.

2.1 RFID Traceable Networks: Preliminaries

2.1.1 RFID Systems

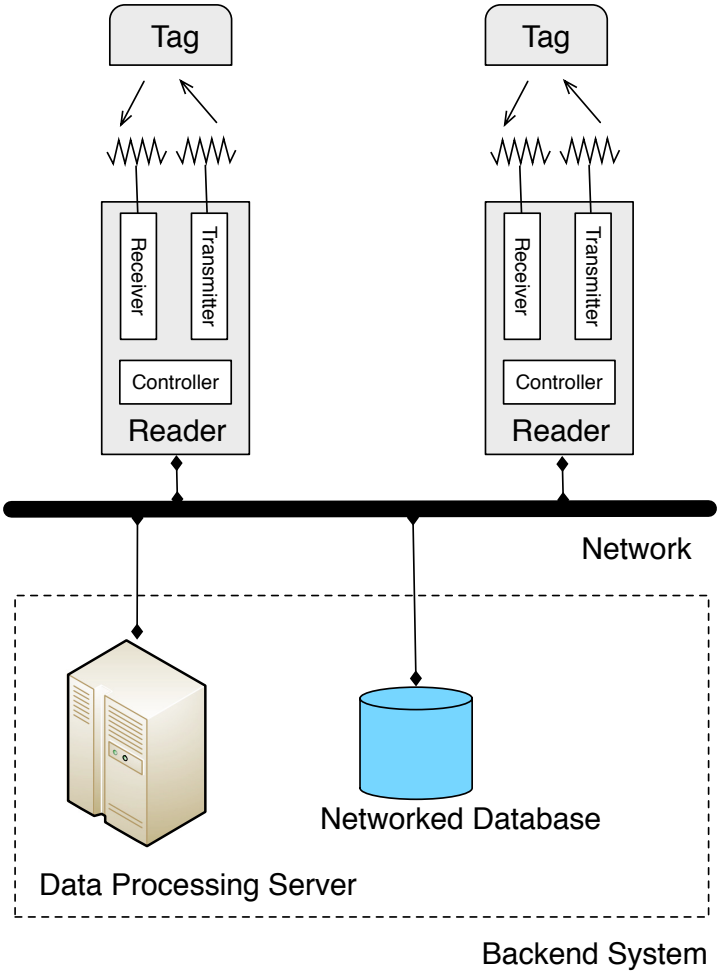


Figure 2.1: Overview of an RFID System

RFID (**R**adio **F**requency **ID**entification) is a technology that transfers the information between an electronic tag and an interrogator using radio waves. It is used to create a seamless link between individual, physical objects and their digital natives. RFID allows individual objects to be uniquely and automatically identified using wireless communications to extract identifiers from RFID tags

attached to objects. In contrast to traditional identification technologies such as magnetic strips or barcodes, RFID is a contact-less technology that operates without line-of-sight restrictions [Fin03].

Regardless of the underlying technologies around which an RFID system is built (e.g., microelectronic tags, surface acoustic wave tags, tags using multiple resonances to encode data and so on), all modern RFID system infrastructures can be categorized into three primary components, namely *tags (labels)*, *readers*, and *backend systems*. Figure 2.1 illustrates the interconnected components of a typical modern RFID infrastructure.

Tags. Tags, also called RFID labels[1], are attached to objects. A tag contains an Integrated Circuit (IC) or a chip that stores the identification of the object to which the tag is attached, and an antenna that communicates the information via radio waves. When a tag passes through an electromagnetic field generated by a reader, the tag communicates to the reader the identification information. Consequently, there is no line-of-sight requirement for object identification in RFID systems.

The data stored on the label, object identification information, may be an **E**lectronic **P**roduct **C**ode (EPC) [SBE01], which is a unique item identification code. Although a variety of existing as well as hitherto undefined identification codes can be encoded as EPC, an EPC typically contains information that identifies the manufacturer, the type of item and the serial number of the item.

RFID tags can be classified based on their frequency of operation (Low Frequency, High Frequency, Ultra High Frequency or Microwave), or according to powering techniques (*passive*, *semi-passive*, and *active*) [Fin03]. An active tag

[1]For consistency, we use the term"tag" throughout this dissertation.

has its own transmitter and a power source to power the microchip's circuitry and broadcast signals to an RFID reader. The power source is either connected to a powered infrastructure or uses energy stored in an on-board battery. In the latter case, an active tag's lifetime is constrained by the battery. A passive tag does not have its own power source and scavenges power from the electromagnetic fields generated by readers. A passive tag also has an indefinite operational life and relies on reflecting back the electromagnetic (EM) field generated by the reader and modulating the reader's EM incident on the antenna to send information back. Semi-passive tags use their own power source to run the microchip's circuitry but scavenge power from the waves sent out by readers to broadcast their signals.

Active and semi-active tags are more expensive and typically used for high-value goods and/or large assets that need to be tracked over long distances. For example, the U.S. Department of Defense uses active tags to track many containers being shipped to bases and units overseas. On the other hand, passive tags are very inexpensive (as cheap as 20 cents) and can even be used for common materials in very large quantities. Currently, significant efforts are being undertaken to achieve *5-cent* tags by shrinking chip size, cutting antenna cost, and increasing tags consumption (e.g., RFID mandates from Wal-Mart and U.S. Department of Defense).

RFID tags appear in a wide variety of shapes (e.g., key fobs, credit cards, capsules, pads), sizes (e.g., small as a grain of rice, big as a six inches ruler), capabilities, and materials. Tags can have metal external antennas, embedded antennas, or printed antennas.

In this dissertation, we will not give a detailed review of physical principles regarding RFID hardware design. Interested readers are referred to [Fin03, Wan06].

Readers. The readers' function is to generate an electromagnetic (EM) field to power tags (when passive tags are employed) and facilitate communication with tags. RFID readers communicate with tags using a radio frequency interface. Either a strong energy storage field near the reader's antenna, or radiating EM waves, establishes the RF interface. Communication between a reader and a tag may involve interrogating the tag to obtain data, writing data to the tag or beaming commands to the tag so as to affect its behavior. The readers consist of their own source of power, processing capability and an antenna (antennas). In addition, most modern RFID readers are equipped embedded systems with networking capabilities (WIFI or LAN) to allow readers to be networked with other computing hardware. Typically, readers are connected to a backend system via the networking interfaces (as outlined in Figure 2.1).

RFID readers are generally placed at fixed locations with their antennas strategically placed to detect tagged items passing through their EM field. RFID readers can read multiple co-located tags simultaneously (e.g., up to several hundred of tags per second). The reading distance ranges from a few centimeters to more than 100 meters, depending on the types of tags, the power of readers, interference from other RF devices and so on [Fin03].

Handheld RFID Reader is one kind of the *Mobile RFID Readers*. Compared with fixed readers, mobile RFID readers are not deployed at fixed location. Contrarily, the readers is embedded in PDAs or mobile phones so that they can be carried around. Mobile readers are very important and useful in mobile payment [OP07]. However, its usage in traceable networks is limited. Consequently, in this dissertation, only fixed readers are under our consideration in the discussions.

Backend Systems. The readers are connected to a computer network in which the data is collected and processed. This network may be limited to a single organization, or it may cross organizational boundaries to enable cooperation and sharing between business partners (e.g., manufacturers, warehouses, and retailers).

2.1.2 Traceable RFID Networks

Networked RFID refers to the system in which "rather than storing object-related data on a tag, tagged objects and their associated metadata can be matched through a task-specific network service" [RDT09]. I.e., an RFID tag only stores an identification, while the mapping between the identification and the other data pertaining to the tag is stored somewhere else, but accessible through the Internet.

Modern business often involves participation of multiple parties. For example, in supply chain management systems (Section 1.2), suppliers, transporters, warehouses and retailers are involved. All of the parties set up their *Network RFID* systems to monitor and management the assets/products. The network formed by connecting these isolated systems through the Internet, is called *RFID Networks*. Furthermore, we call the systems built on RFID networks specifically for tracking and tracing, the *Traceable RFID Networks*. Via traceable RFID networks, we can access not only the metadata for a tag though the network but more useful information such as history of the tag and patterns of object movement in the network.

To ease our discussion and better understand a traceable RFID network and its elements, we propose a generic reference model (Figure 2.2) that is agnostic to various traceability applications by abstracting elements of traceable RFID

networks.

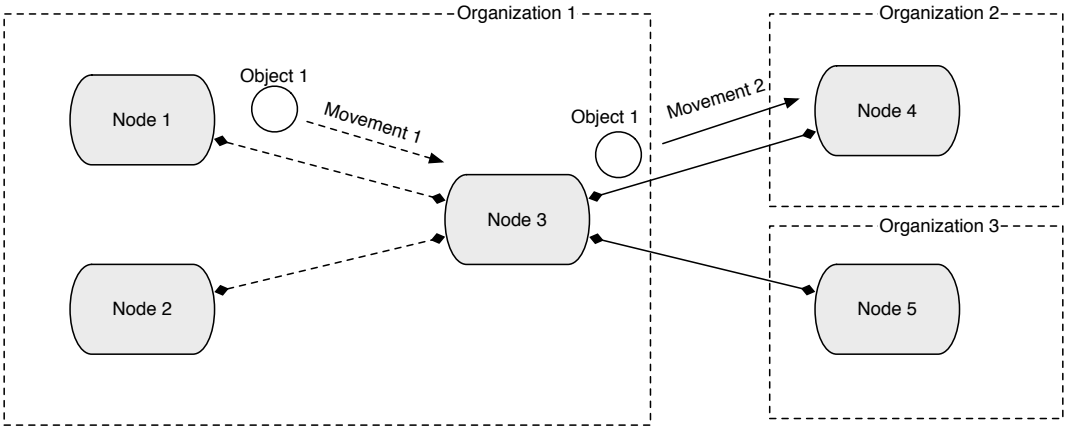


Figure 2.2: Reference Model of Traceable RFID Networks

Figure 2.2 illustrates the following static components in a traceable network:

- **Node.** Nodes represent observation points in a traceable network. There are five nodes in the reference model (Figure 2.2), namely "Node 1" to "Node 5". A node can be a geographic location of an organization or an internal location within an organization. In our discussion, each node has two sets of RFID readers, deployed at the entrance and the exit of the physical location so that the readings of arrival and leaving can be distinguished. But not all the physical locations with RFID reader(s) may formulate as a node in the traceable network. For example, in a supply chain, the internal flow of goods inside a distribution center may not be of any interest to other trading partners while it may be critical for the distribution center to manage its inventory. As a result, for the partners, observation points in the distribution center do not count as *nodes*, while for the distribution center, they do. Whether or not an observation point is treated as a node is determined by user requirements and the context of the

application. In this dissertation, we use v with subscript as the notation for nodes. The set of nodes is noted as $\mathcal{V}$[2].

- **Organization.** Organizations represent the owner of a set of assets and related data. Each organization governs a number of nodes and collects data from them via the RFID readers. The data may be stored in a single database or several databases. Nevertheless, nodes in the network can freely access data from each other. From the view of the global network, each organization is a data source, providing the information about the the arrival and leaving of objects regarding itself, meanwhile, the internal workflows are well encapsulated.

- **Object.** An object represent a tagged item with a globally unique identifier. Objects are noted as o.

- **Connection.** A connection is a link between nodes. It is established statically (e.g., by the partnership of organizations). However, it is *quasi-static* because these partnerships or supply paths may change over time. Each connection may be characterized by several properties or meta data (e.g., distance to neighboring nodes, possible methods and cost of travel). There are two types of connections, namely *Internal Connection* and *External Connection*, illustrated by the dashed line and solid line, respectively. In most cases, external connections are those we are interested in because they represent the business transactions between two organizations. Internal connections represent the internal workflows, which is valuable to the organization only. A connection has two attributes, source node v_s and destination node v_d. Connections are directional, i.e., connection (v_1, v_2)

[2]We choose to use graph theory notation v (for vertex) because n is often used as a measurement.

and connection (v_2, v_1) are different. The set of connections is noted as $\mathcal{E}$.

- **Network.** A network is a set nodes and a set of composite *connections* between them. It represents the direct or indirect relationship between *nodes*. According to data sharing policies, networks are categorized into two types, *Open-Loop* networks and *Closed-Loop* networks. Within a closed-loop network, data is shared by nodes that belong to the same organization. On the other hand, nodes in an open-loop network normally belong to different organizations. Figure 2.2 shows a simple network formed by five nodes and four connections. Node 1, Node 2 and Node 3 form a closed-loop network. A network is represented as a directional graph, where the nodes are the vertexes and the connections are the edges. Network $\mathcal{G}$ is formally defined as $\mathcal{G} := (\mathcal{V}, \mathcal{E})$.

There are two important concepts about the dynamic relationships of objects in a traceable network.

- **Movement.** Movement captures the change of locations of an object, from a source node to a destination node[3]. Similar to connections, movements can be classified as *Internal Movements* and *External Movements*. "Movement 1" illustrates an internal movement, which happens inside an organization, meanwhile, "Movement 2" illustrates an external movement, which is across the boundary of organizations. A movement has four attributes: 1) the source v_s, 2) the destination v_d, 3) the start time t_s and 4) the end time t_d. A movement m is formally defined as $m := (v_s, v_d, t_s, t_d)$.

- **Path.** A set of ordered movements establishes a path p (e.g., "Movement 1" (m_1) and "Movement 2" (m_2) form a two-segment path). The length

[3]For the convenience of discussion, we simply them as "source" and "destination".

Notation	Description	Definition
v	A node.	N/A
o	An object.	N/A
e	A connection.	$e := (v_s, v_d)$
m	A movement.	$m := (v_s, v_d, t_s, t_d)$
p	A path.	$p :=< m_1, m_2, ..., m_n >$ or $p :=< v_1, v_2, ..., v_{n+1} >$
$\mathcal{T}$	The time domain.	N/A
$\mathcal{V}$	The set of all nodes.	$\mathcal{V} := \{v_1, v_2, ..., v_n\}$
$\mathcal{E}$	The set of all connections.	$\mathcal{E} := \{e_1, e_2, ..., e_n\}$
$\mathcal{G}$	A network.	$\mathcal{G} := (\mathcal{V}, \mathcal{E})$
$\mathcal{P}$	The set of all paths.	$\mathcal{P} := \{p_1, p_2, ..., p_n\}$
$\|\mathcal{V}\|$	The number of nodes in $\mathcal{V}$.	The notation itself.
$\|\mathcal{E}\|$	The number of connection in $\mathcal{E}$.	The notation itself.
$\|p\|$	The length of p.	The notation itself.

Table 2.1: Notations in the Reference Model of Traceable Networks

of a path $|p|$ is defined as the number of segments in p. Paths are records about the history of an object in both spatial and temporal dimensions. It should be noted that although path is defined in terms of movements, it can also be represented by connecting the sources and destinations of movements because two conjoint movements in a path must share a same node. For example, in Figure 2.2, the path (m_1, m_2) can also be represented as (v_1, v_3, v_4).

Table 2.1 summarizes the notations that we use throughout this dissertation for the components and relationships defined above.

2.2 Traceability in RFID Networks: Applications, Queries and Requirements

GS1[4], a global organization dedicated to the design and implementation of global standards for supply and demand chains, proposes the definition of traceability as "the ability to trace the history, application or location of that which is under consideration" [GS1] (ISO 9001: 2000). Although the context of [GS1] is based on supply chain management, this definition is appropriately generic for other application areas.

It should be note that GS1 definition only refers to *historical* information. We argue that the ability to establish the present and predict the future state is a significant addition to traceability applications. For example, when an object leaves a location v, the only information recorded is its last observed location (i.e., v). There is a gap in information available about its destination or expected time of arrival. Such information would be potentially useful in making effective business decisions. Consequently, it is useful to articulate the implied meaning of traceability. We formally define the traceability as the following [WRS11]:

Traceability is the ability to retrieve past, present, and potentially, future information about the state (e.g., location) of an object.

Networked RFID systems have the potential to create revolutionary applications by enabling real-time and automatic traceability of individual objects.

2.2.1 RFID Enabled Traceability Applications

The underlying identification technologies predominantly used in existing traceability applications (such as optical barcodes and human readable codes) require

[4] `http://gs1.org`

human operators and are labor intensive for implementation at the individual product level. Printed barcodes are also a line of sight technology, prone to failure by effects that reduce the visibility of the barcode (e.g., dust, dirt, physical tears). There are also other issues such as delays in transactions (e.g., barcodes need to be correctly aligned to be read) and identification inaccuracies due to human operator errors. Consequently, these systems have an important impact on the quality of traceability information.

Research suggests that the process of manually recording a re-usable container number and entering it into a computer before shipment is susceptible to 30% error [Fin03]. The impact of such errors is costly. In manufacturing environments, scanning errors as a result of associating the wrong container to the processing steps can result in the whole batch of products being discarded due to quality assurance reasons. The capabilities of identification technologies and the costs involved to identify products at an instance level have prevented companies from being able to make decisions at the individual product level[5].

However, with traceable RFID networks, object instances can be precisely and automatically monitored, and their life histories can be recorded in real-time. Traceability essentially improves the quality (accuracy and the level of detail) and timeliness of information leading to better decisions at the business and enterprise level. As a result, by exploiting traceable RFID networks' ability to precisely record and trace product movements automatically, there are many emerging advanced application scenarios, such as reducing costs of inventory errors, eliminating shrinkage, fine grain products recalls and anti-counterfeiting.

[5] http://www.scdigest.com/assets/On_Target/09-02-23-1.php

2.2.1.1 Eliminating Inventory Inaccuracies

The discrepancies between actual and recorded inventory in information systems (i.e., inventory inaccuracy) is estimated to be as high as 65% at a major retailer [DRT01]. Despite significant investments by companies to reduce the information gap, the quality of inventory information is still poor and often leads to inefficient supply chains. A significant portion of inventory inaccuracies are related to two execution problems: *transaction errors* and *misplacement errors*.

Transaction errors occur unintentionally during various transactions such as an inventory count, goods receipt check and at the point of sale (e.g., when a variety of potato is recorded as a different kind by the sales staff).

Raman [DRT01] reports that 16% of items at a leading retailer were missing as a result of products being misplaced at various locations in the store, storage or back room. Misplacement errors impact sales. Culprits of misplacement are not just employees but consumers who may pickup items and subsequently place them in other locations. A leading market research and advisory firm IDTechex, estimates that, annually, hospitals lose close to 15% of their assets by value and are unable to locate 15-20% of their assets resulting in additional costs of US$1,900 per nurse[6].

Traceable RFID networks have the ability to reduce transaction errors through automatic capture of individual item level quantities and location information at various process steps. Similarly, misplacement errors can be minimized by analyzing the data gathered from the movements of tagged items, obtained automatically from a network of readers strategically placed along the supply chain and at each business step.

[6] http://www.idtechex.comresearch/articles/rfid/in/healthcare/and/pharmaceutical/applications/00000518.asp

2.2.1.2 Inventory Shrinkage

Inventory shrinkage, as defined by the Efficient Consumer Response (ECR) group[7], refers to the loss of inventory as a consequence of a combination of internal theft (e.g., employees), external theft (e.g., shoplifters), supplier fraud, and administration errors. Shrinkage results in a staggering annual loss of US$33.1 billion for US retailers, Euro 28.9 billion for European retailers and AU$942 million for Australian retailers [AGG]. RFID traceability networks can improve and, in some cases, even eliminate shrinkage due to theft prevention in the supply chains. More importantly, RFID traceability networks provide us the capability to measure shrinkage accurately, which helps to pinpoint the likely causes.

2.2.1.3 Eliminating Wastage and Damage

The cost associated with food wastage is a significant problem for the food industry. For example, perishable fresh products while contributing only 30% to sales constitute 56% of the total wastage at supermarkets [KF08], which represents a significant opportunity for improvement. Many factors contribute to spoilage including unsuitable variations in environmental conditions during transport and handling, excessive dwell time during loading, transport, and unloading. RFID traceability networks can automatically capture movements, dwell times and condition of products, which make it possible for instant checks of freshness and identification of potential causes of spoilage.

[7] http://www.orisgroup.co.uk/blue_book.asp

2.2.1.4 Fine Grain Product Recalls

Food and drug safety is widely regarded as a serious threat to public health globally. RFID traceability networks will ease the task of product recalls by rapidly and accurately locating specific harmful products in the event of problems such as an illness outbreak due to contaminated food. For example, countries are adopting policies and regulations requiring all cattle to be tagged to allow authorities to quickly locate the source of infected cows in the event of an outbreak of mad cow disease [RCT06]. To achieve fine grained recalls, BT Foodnet[8] uses RFID to track products and provides a full audit trail of ingredients along the supply chain. Then only products with bad material need to be recalled, which significantly decreases the wastage.

2.2.1.5 Anti-counterfeiting

The International Anti-Counterfeiting Coalition[9] estimates that US$600 billion of goods, accounting for 5-7% of the world trade, are counterfeit. The impact of counterfeiting is not only limited to manufacturers and brand owners, but has serious consequences for consumers. The World Health Organization estimated that in 2003 between 5-8% of the worldwide trade in pharmaceutical is counterfeit [FDA]. Counterfeit medicines range from products with wrong ingredients, insufficient active ingredients or products with fake packaging to mimic a medication.

There are a variety of existing techniques for product authentication based on optical technologies such as watermarks, holograms, micro printing, and bio-

[8] http://www2.bt.com/static/i/media/pdf/campaigns/consumer_goods/foodnet_broch.pdf

[9] http://www.iacc.org

chemical technology [BO05]. All these technologies have static markers that are generally applied on a uniform scale to a single class of products. However, biochemical marker tests provide the ability to detect markers but they do not generally quantify the marker, thus leaving open avenues of counterfeiting by dilution. Optical technologies no longer present an adequate deterrent due to the reduction in the cost of producing imitated watermarks and holograms.

RFID enabled traceability has the potential to provide a timely and an automatic trace that can verify the existence of a valid chain of custody through a supply chain, which is commonly referred to as providing an electronic pedigree [RC08]. Recent legislation has even pushed industries to consider RFID technology to comply with electronic pedigree laws. For example, some states in the USA have introduced the pedigree laws [Gov] requiring a verifiable record of drug movement through the supply chain at any time. Furthermore, traceability data can be analyzed using machine learning algorithms to detect and report anomalies in supply chains and to alert potential problems or separate counterfeit products from genuine products using copies of genuine product identifiers [IAM09a].

2.2.2 Traceability Queries

It is difficult to determine exact query requirements because they are largely application dependent. A common application oriented classification of traceability queries proposed in [CKS07a] includes: i) *pedigree queries* that reconstruct the complete historical path of an object through a supply chain, ii) *product recall queries* that detect the current location of objects, and iii) *bill-of-material queries* that return information about all the objects with a containment relationship of a specific object. However, this is not an adequate generalization for supporting

traceability applications.

In this dissertation, we formulate a set of fundamental queries that are useful for most traceability applications, which can be used as building blocks to construct more complex queries. It is evident that the information critical to the success of a traceability application is the determination of an item status (identity, precise location, physical status such as perished/damaged/expired etc.) and history of its path throughout the supply chain. The key objective of traceable RFID networks is to enable the discovery of past (trace), present (track) and possibly future information (prediction) of objects. Consequently, tracking, tracing and prediction are the three fundamental types of traceability queries that can be generalized as being adequate for building traceability applications. Tracking refers to a query to find the current state (such as its current location) of an object. Tracing refers to finding the historical states of an object and prediction queries provide a probabilistic view of possible future states of an object (e.g., the most probable node to be visited next).

We summarize and categorize these queries as the following:

Track Queries. A track query supports the retrieval of the current state, such as the location, of an object. The following is a typical example of a track query:

Q1: Where is object o *now*?

Trace Queries. These queries are designed to discover a part of or the whole life history of an object such as movement information. Some typical examples are:

Q2: What nodes did object o travel through before it reached node v_2?

Q3: What is the travel path for object o?

Q4: What is the travel path for object o before it reached node v_2?

Q5: What are the nodes visited by object o after node v_2?

RFID event data is characterized by both spatial and temporal information [WL05a]. Trace queries can be spatially constrained to discover the location of a given object at specified time, or temporally constrained to determine places where the object has been for a particular time period. Some typical examples are given below.

Queries with *spatial constraints*:

Q6: Where was object o on *12th, Dec. 2010*?

Q7: Where has object o been between *12th, Dec. 2010* and *20th, Dec. 2010*?

Queries with *temporal constraints*:

Q8: When was object o seen at node v_2?

Q9: How long did object o dwell at node v_2?

Q10: How long did object o take to move from node v_1 to v_2?

There are a specific set of trace queries aimed at extracting containment relationships between objects such as finding the objects that traveled in a pallet. These are called *containment trace* queries. Some typical examples are:

Q11: What objects were contained in object o on *12th, Dec. 2010*?

Q12: What was the container for object o when it was at node v_1?

Q13: Where was object o packed into object o_2?

Q14: When was object o unpacked from object o_2?

Finally, a set of beneficial trace queries to support business processes and strategic decisions are desirable to data analysis applications (e.g., ERP applications designed to manage the re-ordering or production of goods). As such, in a supply chain, it is useful to know the average time spent by various products in storage at a particular distribution center or on a shelf at a supermarket. These queries are classified as *statistical trace* queries. Some typical examples are:

Q15: How many objects have been sent from node v_1 to node v_2 last year?

Q16: Which node sent node v_1 the maximum number of objects last year?

Q17: Which node received the minimum number of objects from node v_1 in 2010?

Q18: What is the average dwell time of object at node v_1?

Q19: What is the total number of objects seen at node v_1 in 2010?

Prediction Queries. The objective of a prediction query is to estimate the future state of an object. Some typical examples are:

Q20: What is the expected arrival time for object o at node v_1?

Q21: What is the probability that object o will arrive at node v_1 in the next hour?

Q22: What is the expected location of object o after node v_1?

Q23: What is the expected location of object o after five *movements* from node v_1?

2.2.3 Requirements of Traceability Applications

We followed a comprehensive methodology to elicit and analyze the requirements for traceability applications, especially applications in large-scale, collaborative networks. Our approach considers 1) the responses to a survey[10] conducted among potential end-users and vendors in Australia about the requirements for traceability applications; 2) the evaluations of the outcomes of the EU funded BRIDGE project[11], which aims to develop a traceability platform based on identified industrial requirements from enterprises in Europe; and 3) our analysis of existing literature on traceable RFID networks [WL05a, IAM09b, CKS07b, LC08, KBM06, RSZ10, RWN07].

Based on these works, in this section, we identify a set of dimensions for evaluation existing traceability approaches. We consider the dimensions based on system development and data model requirements.

2.2.3.1 System Development Requirements

We consider the following key system development requirements: 1) *scalability*, 2) *heterogeneity*, 3) *support for unique identifier*, 4) *uncertainty management*, 5) *timeliness*, and 6) *security and privacy*. Among them, we consider the scalability and heterogeneity the most important ones in large-scale traceable network.

Scalability. In large-scale RFID applications (e.g., global supply chains), there will be thousands of readers distributed across and within organizations that generate large volumes of data automatically and rapidly. Data volumes can be enormous (e.g., Wal-Mart generates about 7 tera-bytes of data every day if goods

[10] http://cs.adelaide.edu.au/peertrack/collaboration/survey/
[11] http://www.bridge-project.eu/

are tagged at the item level [SLZ08b]). A scalable architecture framework is required to ensure adequate performance of traceability networks as the number of nodes and volume of data increases. A scalable architecture must address the following issues:

- *Data volumes.* Given the large quantities of potential object instance level data, an appropriate solution that does not involve the permanent storage of individual raw data must be found.

- *Integration.* It should be possible to integrate increasing number of nodes into the traceable RFID network without degrading query performance such as timeliness of responses. This is significant since a linear increment of the number of nodes will also linearly increase the number of nodes that must be searched for object related data in a blind search. And a liner increment of the number of nodes will cause exponential increase of the generated data.

Heterogeneity. A traceable RFID network is established by connecting different nodes, which may belong to different organizations, use different hardware and software systems, store the collected data in different formats. In addition, with the rapid development in RFID technologies, new devices may be introduced. Consequently, traceability systems should be agnostic to such heterogeneity and, ideally, be compliant with global standards for interoperability across of organizations and geographies.

Support for Unique Identifier. Given the distributed nature of data collection and storage, there must be a mechanism for associating products with their relevant life-cycle data in networked information systems as well as on products

themselves. This aspect is fundamental to networked RFID systems. The universally unique identifier (UUID) forms the link between an object and its associated information collected and possibly distributed at various organizations and locations. The UUID can then be used to discover and access information associated with it from distributed information resources, similarly to the manner in which web addresses or Uniform Resource Locators (URLs) are used to access information from the Internet. Supporting traceability applications such as targeted product recall and anti-counterfeiting requires that each architecture supports a unique identifier. The scope of the identifier may be defined by the application. However, for managing global traceability applications with a worldwide focus (such as supply chains distributed across countries), a fundamental requirement is the support for a globally unique identifier.

Uncertainty Management. The responses to traceability queries may not be deterministic since the underlying RFID network is limited by the number of discrete observation points (nodes), hardware performance and data sharing issues. Consequently, a significant challenge is to manage uncertainty.

The typical causes for uncertainty are summarized as follows:

- *False Positives.* Data generated by readers is limited in accuracy. RFID readers may report a tag identifier which is not stored on a tag within the reader's EM field. This is called "Ghost Read". Essentially, the reader receives incorrect data which is interpreted by the reader as being valid. False positives result in erroneous data that is difficult for information systems to handle.

- *Missing Events.* A reader may miss identifying an object or a temporal malfunction of a device may cause a systematic error in event generation.

A missed tag reading results in incompleteness of data, because the information stored in the tag, such as the identifier, is not captured by the reader.

- *Nodal Limitations.* At a given time, an object o may be in movement m from node v_s to v_d, or it may have arrived at v_d but not yet to be identified. If we consider a query "Where is o?", the answer might be v_s, according to the data we have collected. This detachment of the digital observation from physical reality affects the accuracy of traceability queries.

Timeliness. Traceable RFID networks are built on the premise that changes in the physical world are reflected by timely changes in information systems. Real-time traceability information is critical for managing distribution operations, rapid product recalls and service/maintenance operations that need to be constantly re-evaluated based on traceability information of tools and technicians. Therefore, an expectation of a traceable RFID network is that the system should be responsive, with the ability to provide timely information.

It is expected that the queries (Section 2.2.2) should work in both *Pull* and *Push* mode. In the pull mode, the front-end system sends the query to the backend to get an answer, meanwhile, in the push mode, the front-end system gets notified when a certain pre-set condition is met. Normally, the push mode is implemented as the Observer pattern [EV95]. However, this pattern requires a centralized mapping from the observed objects to the observer. When the number of the observed objects increases linearly, the time used to notify the observers increases linearly too. This causes a severe delay in large-scale traceable networks.

Security and Privacy. RFID is a pervasive technology capable of mass serialization and unobtrusive scanning from a distance. So no discussion is ever complete without addressing various security and privacy related issues. Traceable RFID networks are susceptible to issues arising from vulnerabilities in RFID technology [Jue06] as well as associated information systems. For example, competitors of an organization (such as a rival supermarket) may scan another organization's inventory labeled with RFID tags or eavesdrop on the organization's own valid operations to obtain valuable information, such as sales data, to ascertain the performance of its competitors (an act commonly referred to as corporate espionage). The fact that a third party can eavesdrop on a conversation between a tag and reader from a distance is a fundamental vulnerability.

There are numerous publications [AF05, Dim05, FDW04, OTY09, JP02] that address vulnerabilities of RFID systems through improved security features such as the kill functionality for Class 1 Generation 2 tags and lightweight security mechanisms suitable for RFID devices. Furthermore, there is a mature and standardized set of cryptographic tools (e.g., public key security mechanisms such as RSA and Elliptic Curve Cryptography, private key mechanisms such as the Advanced Encryption Standard) available for securing computer networks and networked resources. Therefore, we will only consider the traceability system's ability to manage RFID data without violating privacy or compromising security of partner organizations participating in a traceable RFID Network.

2.2.3.2 Data Model Requirements

Most modern RFID readers collect data from tag reading events as a triple tuple {ID, timestamp, node}. The temporal-spatio information is implicit. For example, to discover the dwell time of object o, an infant formula, at node v, a storage

shelf at a distribution center, we have to sort all reads for o at v in time and the dwell time is obtained by the time difference between the first and the last reads. The level of pre-processing required to respond to such a query makes the processing of this query inefficient. In particular, for large-scale systems, it will result in severe performance issues. Most traceability queries are in fact much more complicated.

To make query processing more efficient, a high level data model that considers the characteristics of queries is required. We propose the following requirements for a suitable data model based on the traceability queries discussed in Section 2.2.2:

- **Temporal Abstraction.** RFID data is generated dynamically and associated with timestamps. It is highly desirable for the model to abstract temporal information from the underlying data, such as dwell time, time taken for a movement, and arrival and departure time. Such high-level temporal attributes will vastly improve the processing of trace queries such as Q8 and Q10 by reducing the number of basic queries that needs to be executed as well as by eliminating the time required to process the related responses to derive a final response.

- **Spatial Abstraction.** A node is associated with a location. Many traceability queries are related to discovering movements of objects between nodes (e.g., Q2-Q7). Similar to temporal abstractions, it is desirable for the data model to be able to provide a high level representation that captures object movement, path and other spatial information to efficiently process trace queries.

- **Containment Abstraction.** The data model should be able to encapsu-

late changes in containment relationships as objects move across nodes. In other words, the model should be able to preserve the dynamic relationships between parent and child objects. For example, individual items are packed in cases and pallets, which are then unpacked or repacked in new pallets. With containment relationship captured in the data model, queries Q11-14 can be effectively processed. In addition, the storage cost can be significantly reduced by grouping the records.

- **Statistical Constructs.** To efficiently process statistical trace queries (see Q15-Q19), it is highly desirable for the data model to be able to encapsulate low-level statistical information such as sums and averages based on RFID event data collected by the system. Although this is not an essential attribute, processing a statistical query such as Q16 over a traceability system with a complex supply network structure (multiple pathways into and out of a node) would not be possible using a set of low level trace queries we have discussed in Section 2.2.2.

- **Uncertainty.** We have discussed the need for addressing uncertainty introduced as a result of the imperfections in the physical layer in Section 2.2.3.1. In addition to information systems' support, appropriate data models are required to capture and model the uncertainty in the status of objects derived from observed events and the actual status of objects in the physical world.

A good data model lays the foundation for efficient traceability query processing. However, for the data model to be easily integrated into a traceable RFID network, there are also other relevant considerations that are not directly linked to supporting traceability queries. The most significant one is *storage efficiency*.

RFID data can be voluminous in large-scale applications. Storage requirements become a crucial issue for large quantities of RFID data which need to be accessed rapidly to support real-time performance of traceability applications. Good data compression methods can effectively and efficiently decrease the processing time and storage footprint.

2.3 Overview of Traceability Models for RFID Data

Data models determine the structures of the data storage and representation. Designing appropriate data models significantly affects the performance of the whole system. Due to the nature of large-scale traceability applications, data models must be appropriately designed to support various queries in highly-dynamic, data-intensive environments. In order to support traceability applications in different business contexts, generically fundamental data models are necessary, which are expected to meet the requirements in Section 2.2.3.2.

In this section, we examine a set of representative research work on data modeling and corresponding query processing techniques for traceable RFID networks. We analyze them by considering the requirements in Section 2.2.3.2.

2.3.1 DRER Model

Dynamic **R**elationship **ER** (DRER) [WL05a] (Figure 2.3) is the data model used by siemens' RFID middleware system. It is one of earliest RFID data models. It abstracts the static and dynamic entities including *object*, *reader*, *location* and *transaction*. Interactions are modeled as either state or event based relationships. In [WL05a], low-level temporal and spatial queries are considered. Although the design takes containment into consideration by introducing the *Containment*

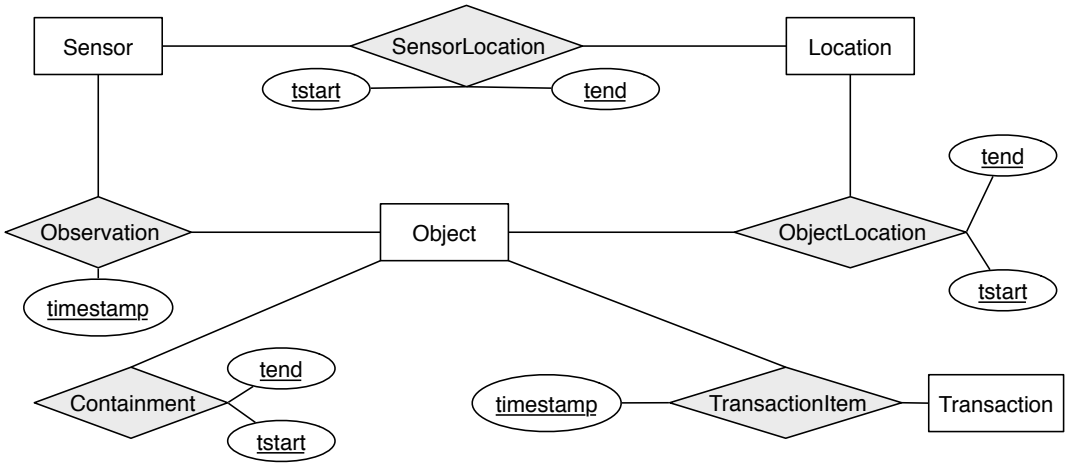


Figure 2.3: RFID Data Model Overview : DRER Model

relationship, it does not mention how this relationship is captured.

DRER models the transition of states for objects using the dynamic relations (e.g., *ObjectLocation*). This model is simple yet expressive that it can be used to answer all the queries that we summarized in Section 2.2.2, either directly or by composing the low-level queries.

The simplicity of this model brings some shortcomings:

- The model and the query processing techniques assumes all data are stored in a single database or database cluster. It is hard to be used in distributed, heterogeneous systems.
- The model focuses on modeling individual RFID events. It lacks support for complicated data structures, such as path. Consequently, some queries have to be implemented by composing low-level queries which is not efficient.
- The model lacks support for statistical queries.

2.3.2 RFID Cuboid

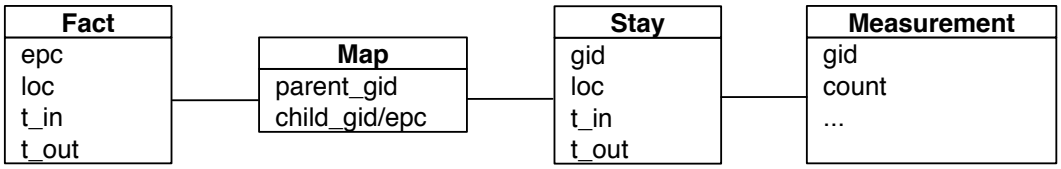


Figure 2.4: RFID Data Model Overview : RFID Cuboid

The idea of *RFID Cuboid* [GHL06a] is based on the observation that individual objects tend to move and stay together (i.e., bulky object movements). The records for the objects moving along the same segments can be merged without loss of information. The term "cuboid" implies that data is merged at some point. Compared with DRER, RFID Cuboid is a data mining model instead of dynamic event-driven model. Figure 2.4 shows the key tables for the RFID-Cuboid. The *Fact* table is essentially the same as the *ObjectLocation* table in DRER. RFID Cuboid introduces the *Stay* and *Map* tables for data compression and measurement. The main idea of compression is that the *Map* table records the mapping for high-level group to its child groups or objects for each segment in the movements, meanwhile the *Stay* and *Measurement* tables record the related data for the groups, instead of each individual object.

The most important advantage of RFID-cuboid is the efficient support of statistical and path-oriented queries, by grouping the objects and materialization of the group information. This materialization significantly improves the performance of query processing. However, the storage used by RFID-Cuboid is more than that by DRER because of the additional tables. This additional storage cost is further reduced by the same authors' recent work by introducing a *Gateway-Based Movement Graph* model [GHC10]. This enhanced work makes an assumption that there are some "gateway" nodes in an RFID network, which

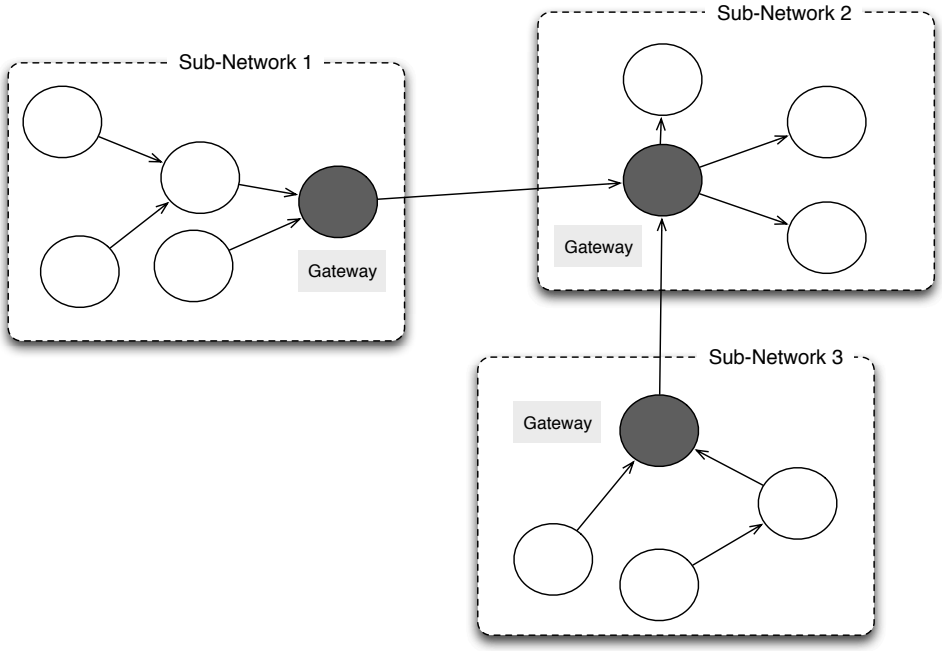


Figure 2.5: RFID Data Model Overview : Gateway-based RFID Cuboid

have either high fan-in or high fan-out edges as illustrated in Figure 2.5. These gateway nodes connect the sub-graphs together.

The RFID Cuboid can be established around the gateway nodes, and de-grouped within the sub-graphs. Instead of using the starting location of a group of objects as the root, the gateway-based movement graph selects the gateway nodes as the root. In this way, the root groups are the largest, so that the number of groups are minimum. As a result, the size of *Stay* and *Map* tables are further reduced. This is very useful for large-scale, distributed traceability applications (e.g., global supply chain systems).

The RFID-Cuboid model efficiently compresses the data and improves the performance of queries using a tree-structure. However, it is highly dependent on the data distribution. The performance is significantly affected if objects do not exhibit bulky movements. Consequently, this data model is only suitable for

large datasets that share some common properties (e.g., move together in bulky mode).

2.3.3 KAIST Trace Model

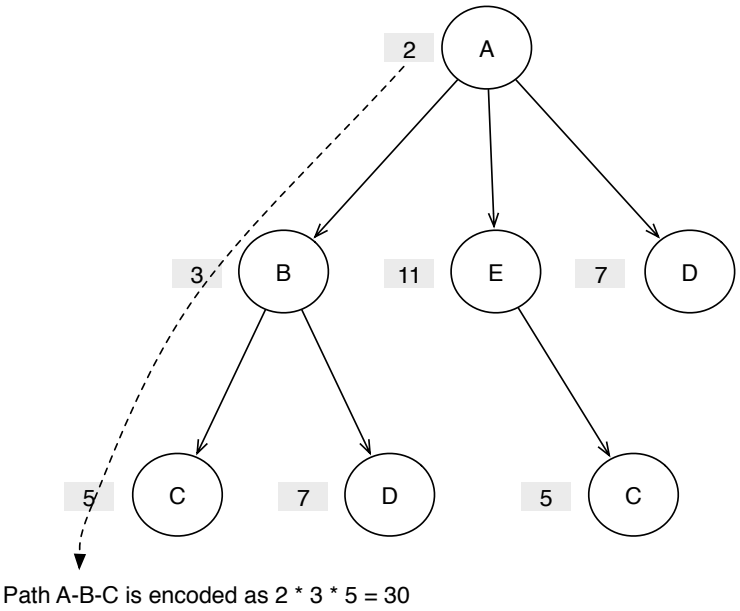


Figure 2.6: RFID Data Model Overview : KAIST Trace Model

The researchers from KAIST (Korea Advanced Institute of Science and Technology) proposed a novel model to efficiently encode and query path information in an RFID database [LC08] [12]. The encoding scheme is based on the Chinese Remainder Theorem and can encode a path to a serial number level. A query processing language is also proposed. The idea of this model is to represent the paths as a forest. Each starting location is presented by the root of a tree and receiving locations are child nodes of sending locations. Figure 2.6 illustrates the model by an example.

[12]The authors did not give a name to the model. For the convenience of discussion, we call it "KAIST Trace Model".

Each node in the network is marked by a unique prime number. Thus, each path can be encoded by a number namely *Element List Encoding Number*, which is the multiplication of the prime numbers from the root to the leaf node. For example, the path $A \rightarrow B \rightarrow C$ is encoded as $2 * 3 * 5 = 30$. Moreover, according to *Chinese Remainder Theorem*, an *Order Encoding Number* is introduced to calculate the order of a node in a path for decoding. Interested readers are referred to [LC08] for more details.

Experiments have proved that this encoding scheme with the query processing method efficiently discovers the path information for a given object. In particular, for most queries, the KAIST model is better than RFID Cuboid. In addition, similar to the RFID-Cuboid model, it significantly decreases the data storage size. However, this model does not assume that the object moves in groups, so it can be used in more scenarios. The KAIST trace model is path-centric and can efficiently process path-oriented queries.

KAIST trace model has several issues which prevent it from being used in distributed environments. Firstly, the encoding/decoding scheme is path-oriented, thus data from different nodes must be stored in the same sever, otherwise it is not difficult to build the tree. Secondly, once the tree is built, any change in the network requires to rebuild the whole tree. In dynamic distributed environments, this process is costly in both time and bandwidth.

2.3.4 SPIRE Model

None of the aforementioned models discuss how the containment relationship is captured. How to automatically infer the containment relationship is still an open problem. It is a multiple-layer problem which involves hardware configuration, data cleansing, uncertain data management and other techniques. *SPIRE* is a

representative work on this problem.

An earlier work from SPIRE [Coc07] proposes two options to detect containment relationships. One option is the manual approach, the other is to configure RFID readers so that it can read only the most outer container's tag. However, it does not solve the regrouping problem. In addition, due to the unreliability of RFID readers, it will not work perfectly even with the assumption that no regrouping exists. In a recent work by the same authors [CTD08], the SPIRE system is improved to detect containment relationship using a statistical method. In this approach, the containment is inferred by the historical co-location of tags. A time-varying colored graph model is proposed as shown in Figure 2.7.

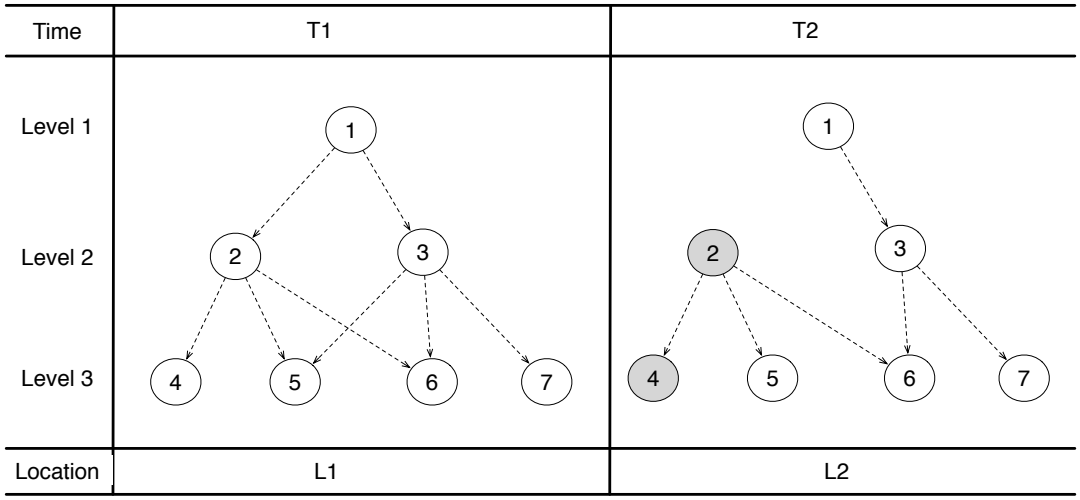


Figure 2.7: RFID Data Model Overview : SPIRE Model

The edges indicate *possible* containment relationship, while the objects detected together at the same location are marked with the same color. At the beginning, the edges are added from the higher level container to lower level objects/containers if they are at the same location. After they move to a new location, some edges are removed if the co-location relationship does not exhibit any more. Ideally, after some point, there should be exactly one path from the

root to a certain leaf. However, this rarely happens because of regrouping. To solve this problem, a probabilistic inference method is proposed in SPIRE. The basic idea is to assign weights to the co-location records and recent record gets higher weight. The incoming edges to a node are sorted by the weighted sums of the co-location records. SPIRE chooses the edge with highest sum to update the containment relationship.

However, the containment relationship is inferred by co-location of tags but it cannot distinguish between co-location and containment. To solve this problem, it is necessary to encode the containment level information in the tag. This makes this model inflexible and expensive, because the tag must be unified across all organizations.

2.3.5 Comparison and Open Issues

Underlying data models play an important role in shaping the higher level architectures for RFID traceability networks. A well-designed data model can significantly improve system performance and decrease persistent data storage requirements. We compare the aforementioned data models using the requirements identified in Section 2.2.3.2, as well as the traceability queries supported by each data model. Table 2.2 and Table 2.3 summarize the results.

From the tables we can see that significant work remains in RFID data models and traceability query processing:

- *Distributed data model.* Most RFID traceability applications are distributed and spread across organizations. It is difficult to assume or require data to be stored in centralized databases. Distributed data models therefore need to be carefully designed to support traceability queries. However, at the

time of writing, to the best of our knowledge, there are no existing data models that meet all the requirements we have outlined. We believe that extensive researches are needed for modeling distributed RFID data.

- *Statistical queries over paths.* Realizing these queries can provide data flow statistics through particular nodes or paths, which is vital for high-level business decisions in traceability applications. Unfortunately, these statistical queries are not well supported by existing data models.

- *Containment queries.* Containment queries over object paths are also important, especially in product recalls where an object (e.g., tainted pork) from a node should be recalled. For these scenarios, it is necessary to find all other objects (e.g., other pork that traveled in the same pallet) that have a containment relationship with the object in question, obtain their paths and recall them. Unfortunately, containment queries are also not supported by most of the existing data models.

- *Uncertainty.* Most existing data models do not take uncertainty of RFID data into consideration. However, as we have discussed in Section 2.2.3.2, uncertainty should be treated as first class citizen in RFID traceability networks.

2.4 Overview of Architectures for Traceable RFID Networks

In the past decade, the rapid deployment of RFID technology is making the collection, processing, integration and sharing of RFID data an active area of

	Temporal Abstractions	**Spatial Abstractions**	**Statistical Constructs**	**Containment Relationships**	**Uncertainty**
DRER	Transaction entity	Location entity	Not addressed explicitly	Observation event	Not addressed
RFID-Cuboid	Stay table	Stay table	Data mining queries	Not addressed explicitly	Not addressed
KAIST	Time table	Path table and path encoding scheme	Partially addressed	Not addressed	Not addressed
SPIRE	Time stamps	Object-Containment Graph	Not mentioned	Object-Containment Graph	Not addressed

Table 2.2: Comparison: Data Models vs. Data Model Requirements

	Track	**Trace (Temporal Constraints)**	**Trace (Spatial Constraints)**	**Trace (Statistical)**	**Trace (Containment)**
DRER	Partially supported (path related queries not supported)	All supported	All support	Not supported directly*	Partially Supported
RFID-Cuboid	All supported	All supported	All supported	Supported	No
KAIST	All supported	All supported	All supported	Not supported directly	No
SPIRE	Not supported directly	All supported	All supported	Not supported directly	Supported

* "Not supported directly" implies that these queries can only be answered by composing low level queries.

Table 2.3: Comparison: Data Models vs. Supporting Traceability Queries

research and development [SLZ08b]. In this section, we first introduce the EPCglobal standards for traceable network. Then we present an overview of current efforts being developed to achieve traceable RFID networks. We study these approaches and compare them using the system development requirements outlined in Section 2.2.3.1. Based on the analysis, we also point out some challenges and open issues that need to be addressed.

2.4.1 EPCglobal Architecture Framework

EPCglobal [EPC] is an organization focusing on developing standards to support RFID in information rich trading networks. The major standards in **E**PCgloal **F**ramework **A**rchitecture (EAF) are illustrated in Figure 2.8. EAF is widely regarded as one of the most well-known RFID network architectures. EAF is a collection of standards for hardware, software and data interfaces, together with several core services (i.e., "EPC Network Services") as shown in Figure 2.8. This framework is a layered architecture that separates functionalities into three isolated modules, namely *identity*, *capture*, and *exchange*.

Identity The identity layer standardizes data representation in RFID tags (i.e., "Reader Air Interface and EPC Spec"). An important standard in this layer is the Electronic Product Code (EPC, not shown in the figure). EPC is designed to be a scalable license-plate identification number that enables linking between an individual product and its associated information resources or backend information services. The air interface standards define the specifications for data and commands to be transferred between tags and readers (e.g., the Class 1 Generation 2 UHF Air Interface Protocol Standard or "Gen 2"[13]).

[13] `http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2`

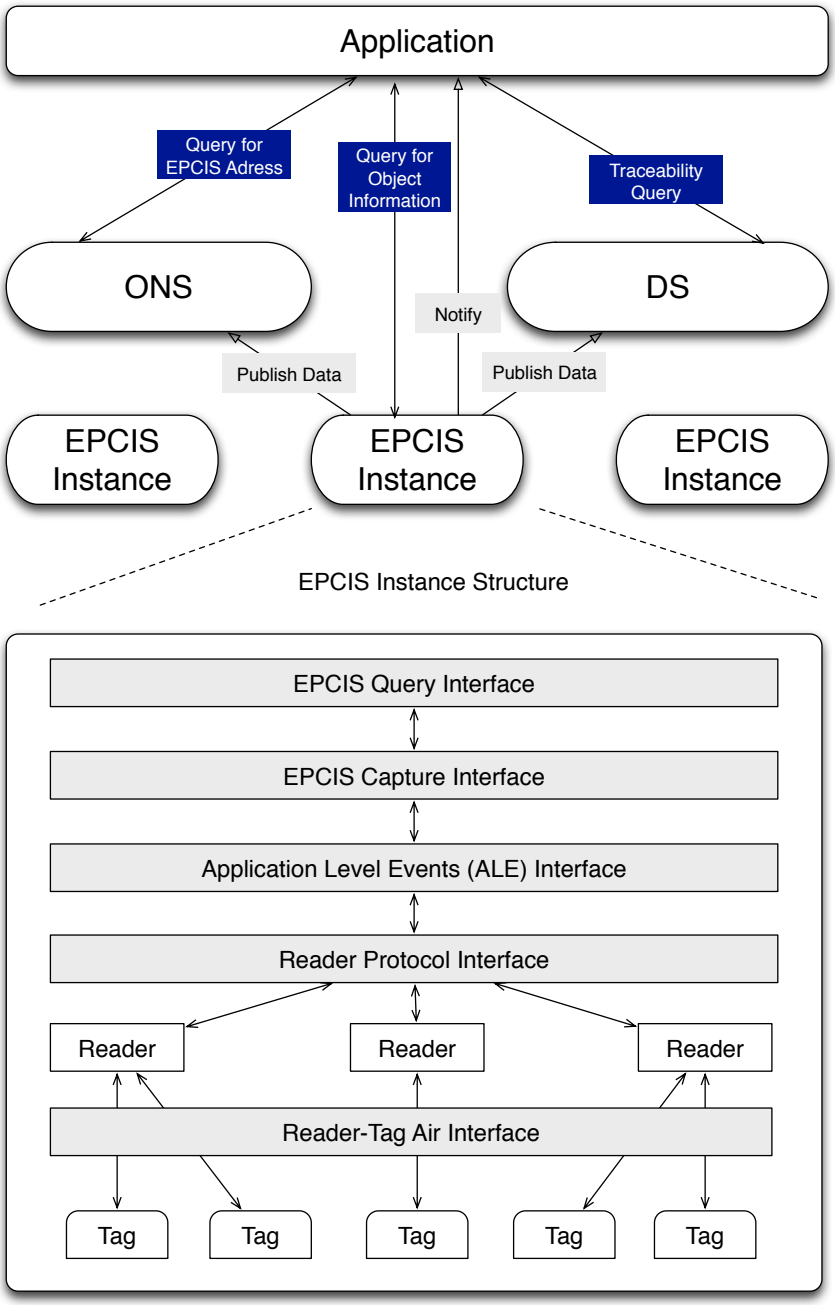


Figure 2.8: EPCglobal Architecture Framework

Capture The capture layer consists of standards for reader management, reader protocols and most importantly, the Application Level Event (ALE) interface.

ALE is a common interface for accessing processed RFID data and controlling the collection of raw RFID event data sent from RFID readers. The ALE specification describes the behavior of aggregating and filtering of RFID data within a period of time. This period of time is called *Event Cycle* (EC). At the end of an event cycle, the data collected is processed and transformed into a report with filtered event data containing *what*, *when* and *where* information. ALE also defines these report specifications.

Exchange The data exchange layer is designed as a service-oriented architecture [PTD07]. In this layer, there are three core services defined, namely *EPC Information Service* (EPCIS), *Object Naming Service* (ONS) and *Discovery Service* (DS):

EPCIS is the first step to enable data sharing and object tracing between partners. It defines a set of interfaces for data capture and query. It also defines a high-level data model which classifies data as either Master Data (static) or Event Data (dynamic). The cooperation and data sharing methods are achieved by two interfaces, namely *EPCIS Query Control Interface* and *EPCIS Query Callback Interface*. Both interfaces define access control policies to allow only authorized trading partners to access data.

ONS functions like Domain Naming Service (DNS) in Internet protocol stack. It uses the ID of an object to retrieve the address of the EPCIS containing its data. It is based on DNS and uses a particular type of DNS record, called Naming Authority Pointer (NAPTR) record[14], to provide for future flexibility. It should be noted that, ONS only resolves the address of the EPCIS instance where the

[14]IETF RFC 2915

EPC is originally assigned to the object, although the object may be scanned and stored in other EPCIS instances.

DS is currently under development[15]. This service is expected to discover information which may distributed across many EPCIS instances, for a specific object.

2.4.2 BRIDGE

Building **R**adio frequency **ID**entification for the **G**lobal **E**nvironment (BRIDGE)[16] is an European Union funded project to develop networked RFID systems. Although BRIDGE utilizes the EPCglobal standards, it explores many related fields including hardware, software and security with extensions. Work carried out within the BRIDGE project includes the implementation of the EAF (see Section 2.4.1), development of prototype Discovery Services, definition of essential interfaces such as DS publish interface, and development of algorithms and tools for building traceability applications. The BRIDGE project has explicitly taken track and trace into consideration and designed specific services for traceability queries while taking into account uncertainties. A number of successful industrial trials have been achieved in the project[17].

To support the Serial Level Lookup Service, BRIDGE has leveraged and extended the EPCglobal standards by developing a *DS Query and Publisher* interface along with the development of Discovery Services . The BRIDGE project has addressed an existing gap in the EAF by developing the key services necessary to enable traceability applications[18].

[15] `http://www.gs1.org/gsmp/kc/epcglobal/discovery`
[16] `http://bridge-project.eu`
[17] `http://bridge-project.eu/index.php/bridge-public-deliverables/en/`
[18] Researchers from IBM Almaden Research Center have also developed prototype

An important part of BRIDGE is a *Supply Chain Node Network Hierarchy Model* that encapsulates a supply chain model for capturing physical flow of objects in terms of nodes and connections. The track and trace model is used to model the actual state of an object and the observed state reported by the RFID devices based on a hidden Markov model (HMM), which distinguishes between the actual and the latest observed state. The HMM model describes the uncertainty under which the observed state reflects the actual successive states of the object. The *Tracking Algorithms* which consist of probabilistic and non-probabilistic algorithms provide procedures for track, trace, and prediction queries. Probabilistic algorithms are particularly designed to address uncertainty of reported observations (e.g., missing reads).

Architecture extensions developed within BRIDGE certainly overcome some limitations of the EPC Network (e.g., uncertainty, discovery services and supporting interfaces) and meets all the requirements already satisfied by the EPCglobal Architecture Framework (e.g., explicit support for a unique identifier, heterogeneity, security and privacy). Security is even further strengthened in BRIDGE through strict access control policies governing published records as well as parties authorized to publish to DS. This is beneficial to prevent corporate espionage and surreptitious collection of business sensitive information.

However there are still some limitations in BRIDGE:

- *Flexibility.* The track and trace model based on HMM overcomes uncertainties. This is handled in BRIDGE based on a *static* business model and a learning phase used to establish the network parameters such as transition probabilities for objects that move across a network of nodes. Consequently

DS [RKB06]. However, the BRIDGE project in particular has demonstrated the use of DS along with the EPCglobal architecture in various industrial projects

the model is hard to adapt to dynamic changes in the physical world (e.g., additions or removals of supply network nodes).

- *Scalability.* The Discovery Services model has been selected specifically to allow parties on a traceable RFID network to exercise fine grain control over data related objects. However, the need for access control policies for individual object instances and the volume of potential records for the DS to provide an adequate level of support for serial level (item level) traceability queries raises concerns regarding the scalability of the approach. In particular, the access control policies are complex to manage given multiple trading parties and billions of units of items moving through a network.

- *Data Model.* BRIDGE defines additional data model, the Supply Chain Node Network Hierarchy model, on top of EAF data model. By encapsulating prior knowledge about the supply network, this model does not maintain dependencies of nodes. This might be a significant problem since all supply chain partners must maintain an individual supply network model and communicate physical changes to all other parties in an off-line manner. Such an unprecedented degree of collaboration may not be desired by businesses in practice.

- *Traceability Queries.* The track and trace model and the supply chain network structure model are used to answer track, trace and prediction queries. More specifically, BRIDGE supports track queries, trace queries with spatial and temporal constraints, containment trace queries and prediction queries. Other types of queries (e.g, statistical trace queries) are not supported explicitly. Execution of these unsupported queries must be implemented at the application level by aggregating data through low level trace queries.

2.4.3 IBM Theseos

IBM's Theseos [CKS07a] is a query engine capable of processing complex queries across organizations to enable the development of traceability applications in a completely distributed setting. Theseos relies on a novel traceability data model that eliminates any data dependencies between organizations, which serves as a global schema that allows the formulation of a query without knowledge on how the data is stored or where it is located, and how a tracking query is executed [ACK06a]. In particular, Theseos introduces two attributes in its data model, namely *sentTo* and *receivedFrom*, that each organization is required to maintain for the movement path of an object. With this information, it is possible to minimize the number of nodes to be visited without flooding queries to all nodes in the network. Traceability queries are first processed locally. Based on the outcome of this process, the query is further analyzed. It may be rewritten and then forwarded to other distributed databases. The results retrieved from the network are added to the local results and post-processed are required to yield the final response.

One advantage of Theseos is that the data is not centrally maintained and each organization has the ability to selectively share traceability data with other trading partners. Another advantage is its scalability. Since data is processed and stored in each individual node, the workload is naturally distributed. Unfortunately, to obtain the *sentTo* and *receivedFrom* information, Theseos requires high synchronization with other enterprise data (e.g., billing or accounting information). This is impractical for many applications where such enterprise data may be unavailable. Another significant disadvantage of Theseos is its instability. If any of the peers is down, all queries relevant to that peer will fail. This is because of the difference between peer-based RFID solutions and other peer-based

data sharing applications such as Bittorrent [WFC05]. Bittorrent allows redundancy to exist and makes good use of this feature to increase data availability and reliability. But peer-based RFID solutions keep data strictly private at each node (i.e., there is no redundancy).

Theseos allows enterprises to selectively control access to traceability data using Hippocratic Database (HDB) [AKS02] technology based on ten principles rooted in privacy regulations. Consequently, successful execution of a traceability query requires the inquiring party having the access privileges to the data stored at the nodes along the movement paths of an object.

2.4.4 DIALOG

Distributed Information Architectures for cOllaborative loGistics (DIALOG) [DIA, FN09] is a collection of projects focusing on developing a distributed information sharing system. The DIALOG system is an open-source solution built on a P2P architecture for tracking objects using the DIALOG middleware system. Similarly to Theseos, DIALOG stores the data at places where it is collected. However, DIALOG requires the manufacturer of the tagged goods to maintain a server (software agent) which records the movement of the objects. Each *Information Provider* node notifies the specific DIALOG agent when an object is identified by its ID@URI identifier which consists of two components, a *unique ID string* and a *URI* (Uniform Resource Identifier) where the DIALOG Software Agent of the object resides.

Most of the recent developments in DIALOG have been achieved through an EU funded project named PROMISE[19] (**P**roduct **L**ifecycle **M**anagement and **I**nformation **T**racking using **S**mart **E**mbedded **S**ystems). The central advantage

[19] `http://www.promise.no/`

of DIALOG lies in its open architecture where messages are automatically routed using existing DNS infrastructure. However, using DIALOG with the ID@URI product instance identifiers still does not address the issues of re-routing when URLs change and the relatively long identifiers of the DIALOG system. Consequently, the questions of what product instance identifiers should be used and how to lookup or discover information sources about those instances are still largely an open issue.

The requirement that the manufacturer has to maintain the history information about an object makes the system inflexible. Although DIALOG claims to be P2P, the manufacturer-centric architecture is in fact centralized. As a result, the scalability cannot be guaranteed.

2.4.5 Hierarchical P2P-based RFID Code Resolution Network

In a work presented in [ZLL09, ZLZ09], the authors propose a hierarchical peer-to-peer (P2P) architecture to solve the load balancing and single node failure problems in ONS and DS type resolution infrastructure we have discussed in Section 2.4.1 and 2.4.2. The resolution infrastructure is divided into groups and each group has a super node and the collection of super nodes form a higher level group (i.e., super node group). All groups are organized as P2P networks with unique node identifiers. The super node of each group is usually maintained by the resolution service provider and the load in the group is distributed to the member nodes by its super node.

The objects flowing through a supply chain network are assigned to groups according to the prefix of their EPC. The prefix selected is segmented into two portions C_1 and C_2 where C_1 is the manager number section of an EPC and C_2 segment corresponds to product type code and serial number portions of an EPC

identifier. Group nodes maintain the mapping information between individual EPCs and services endpoints responsible for each EPC such as the URL of an EPCIS. Super nodes are responsible for both starting a query and returning the query result.

The proposed dynamic resolution network in [ZLL09] based on the static network described above is a hierarchical P2P solution for implementing a DS type infrastructure to record object movements and support track and trace type queries. Every event in a local EPC Network should be registered within the group nodes in the form of $< EPC, EPCISAddress{+}event\_type{+}timestamp >$. Upon notification from a group node of the low level index, the super node of the group, S_i, creates a secondary index in the from of $< EPC, v_i\_IP\_Address >$. Finally, based on the P2P protocol used, one or more super nodes, for example v_k, are chosen by the group super node v_i to store the secondary index record in the form of $< Hash(EPC), v_i\_IP\_Address >$. It is then recorded in the secondary index table of super node v_k. Since each object has a unique EPC, all records related to a specific EPC generate the same secondary index hash values as the object moves through a supply chain network. Consequently, all the records related to a particular EPC is managed by a specific set of super nodes with pointers to group nodes that contain mappings between EPCs and service endpoints that observed the movement of the the object.

The architectural extensions only provide a means to enable track and trace queries given an underlying EPC Network architecture. Hence there is no efficient mechanism for realizing high-level track and trace queries. Also, this P2P approach will result in longer and indefinite response times.

2.4.6 Comparison and Open Issues

The aforementioned architectures are compared using the traceability requirements presented in Section 2.2.3.1. Table 2.4 summarizes the results. From the table we can see that despite recent progress in RFID traceability research, many issues still remain to be resolved at the system architecture level in RFID traceability networks.

Unique Identifier Tracking, tracing and predicting the state of individual objects require an unique identification scheme that can be manipulated by information systems. EPCglobal Network, BRIDGE and the Hierarchical P2P based architectures are all built upon EPC. DIALOG relies on ID@URI approach. The cost of guaranteeing the uniqueness of the EPC's company identifier (manager number) requires subscription to EPCglobal[20]. ID@URI approach relies on each company owning a unique domain name. There are three significant differences with the two approaches: i) the cost of domain name registration is nominal while subscription to EPCglobal is more expensive, ii) as a result of using URIs, the DIALOG architecture requires relatively more expensive RFID tags (rewritable) compared to write-once RFID tags used with the EPCglobal approach, and iii) EPCs are centrally managed by EPCglobal, which guarantees the uniqueness with a global scope. The URI used in the DIALOG system is usually a URL, which is quite fragile. For example, if the URL or more specifically the local path of the software agent is changed, objects whose tags have been written with the URL might fail to resolve on the DIALOG network.

[20]No subscription is required for use of USDOD-64 and USDOD-96 EPC identifier types designated for North American military use.

Uncertainty and Prediction Query Support Most of the existing architectures do not address uncertainty explicitly. The assumption that the underlying data capturing technologies are perfect is not correct. Uncertainty in captured data significantly affects the results generated by traceability queries. In recent years, uncertainty has become an active research topic [ABS06, CSP05]. Only architecture extension supported by BRIDGE has provided high-level models and algorithms capable of modeling the uncertainty. However, an important issue with the approach is the need of supervised learning for the models to be useful. Significant work is needed to i) investigate other modeling techniques such as conditional random field (CRF), skip chain CRF [SM07] and Emerging Patterns [GWT09] and ii) consider more dynamic models that does not require a learning phase [RCG10, HFS08].

Scalability The existing architectures have achieved scalability based on either federated or P2P architectures. EPCglobal Network and BRIDGE are federated. The problem with this approach is that the Discovery Service (Serial Lookup Service in BRIDGE) becomes a bottleneck. This issue is considered by the Hierarchical P2P approach by implementing Discovery Service in a pure distributed manner (refer to Section 2.4.5) and by DIALOG (refer to Section 2.4.4) based on its multi-agent design. In general, in a P2P based approach, each node delegate a query to its neighbors if it cannot answer the query itself. For example, in Theseos, queries are processed locally and re-written before forwarded to the next node. However, a significant issue for P2P based approach is that each node must take an equal equity in the network and be open to the idea of having its data stored on different peers that may be controlled by competitive businesses.

Heterogeneity Most of the existing architectures follow the EPCIS standards in the EPCglobal Architecture Framework so that the backend system implementations do not affect the high-level architecture. Perhaps one of the most important distinctions is that EPCglobal Network provides a layered architecture stack with well defined standard interfaces. In contrast, DIALOG does not provide standard interfaces such as the EPCIS for exchanging object related data. Use of standards is critical when dealing with data exchange between multiple organizations and heterogeneous environments. Standards also play a role in enhancing competition in the technology and systems provider market by allowing multiple vendors to compete for system components and technologies.

Timely Response EPC Network, BRIDGE and DIALOG use subscription mechanisms ("push" based approaches) to support real-time data requirements. However in query processing, P2P architectures are not capable of providing time constraints although this is an important requirement for traceability applications. For P2P architectures, a query may be propagated several times and this significantly increases the processing time. Improving the timeliness of query responses is a significant issue and we encourage more research in this direction.

	UUID Support	Scalability	Heterogeneity	Security and Privacy	Timely Response	Uncertainty
BRIDGE	Uses EPC managed by EPCglobal	Similar to the EPCglobal Network but with DS implementations based on the Directory of Resources model [CGP10]. DS may be a bottleneck due to complex security policies required to manage individual EPC records	Uses EPCglobal standards but new interfaces developed are yet to be standardized	Directory of Resources model and access control policies for individual EPC records ensure that any DS client will have to pass two access controls: i) at the DS to obtain the list of EPCIS links and ii) at the subsequent query to the EPCIS to obtain detailed event information)	Subscribing mechanism for real-time responses	Partially addressed through probabilistic algorithms and supply chain network data model. Algorithms are based on hidden Markov models (HMM) and are available to support applications through APIs
IBM Theseos	Able to use existing UIDs including EPC	Distributed and P2P based architecture, highly scalable	Vendor neutral standards based on EPCglobal standards	A peer controls access to its own data using fine grained access controls based on Hippocratic DB technology	Cannot guarantee real-time responses because of P2P query latency	Not addressed explicitly
DIALOG	Uses the ID@URI identifier. Relies on each company owning a unique domain name (however costs are minimal)	Multi-agent based distributed system. Single agent for a product class or type may be a bottleneck	No DIALOG specific standards. However, DIALOG agents support implementation of any data exchange standard such as PMI in PROMISE. Although ID@URI approach is flexible, URLs in the identifier are fragile and inflexible	Data source (e.g., manufacturers) nominated by ID@URI have control over all the data about objects and partner nodes need to update the data source. Other participants therefore lack control on the data	Extensions made to support real-time response through call-back and pull-based mechanisms	Not addressed explicitly
Hierarchical P2P Network	Uses EPC managed by EPCglobal	Distributed and P2P based, highly scalable	Uses vendor neutral standards developed by EPCglobal	Defined security mechanism for communications between peers and access to data	Real-time responses cannot be guaranteed because of P2P query latency	Not addressed explicitly

Table 2.4: Comparison: System Architectures vs. System Development Requirements

Security and Privacy To enable traceability in a distributed RFID system, there must be some level of data sharing between nodes. Access control in DIALOG is shifted away from parties down the supply chain towards manufacturers. Manufacturers of objects exercise dominant control over collecting information from other parties and sharing of that information with client applications. In contrast, the EPCglobal Network (through the Discovery Services mechanism developed in BRIDGE) allows highly granular access control policies to be specified by parties collecting information in order to determine access rights by other entities to product related data. There are trade-offs between the privacy and data sharing. For example, P2P architectures protect privacy adequately by providing nodes with ownership of the data and the choice to respond only queries from desirable parties, which also means some constraints for data sharing. Thus techniques for dealing with the tradeoff between privacy and data sharing is still a research challenge.

2.5 Summary

During the past decade, RFID technologies have developed rapidly and are increasingly used in large-scale, mainstream applications. Traceability is a critical aspect of majority of these RFID applications. Enabling traceability using networked RFID systems brings some fundamental research and development issues. Most of these challenges are due to the large volume of data generated from different organizations, the unwillingness of participants to share data, and data quality issues that arise as a result of the physical layer. Consequently, we need to propose novel solutions to make RFID traceability applications scalable, robust, and secure. In particular, a well-defined data model that takes into consideration unique characteristics of RFID traceability applications such as temporal and

spatial characteristics of RFID data, uncertainty and containment relationships that exist between objects.

We overview the current approaches to enable traceability in RFID networks. By comparing and analyzing these architectures and data models, we have concluded that the area of RFID traceability networks presents many interesting challenges and opportunities that need to be resolved before global traceable RFID networks can be fully realized. In the remaining sections of this dissertation, we propose novel data models and architectures for scalable large-scale traceable RFID networks. Our solutions address most of the requirements proposed in this chapter.

Chapter 3

Peer-to-Peer Model and Architecture for Traceable Networks

Data models are very important for realizing efficient, effective and scalable traceable network. On the one hand, as we have summarized in Chapter 2, existing traceability models are mostly best-suited in centralized database. The data schemas cannot be easily adopted in distributed networks because they either require synchronization with enterprise database, or are static. On the other hand, traceable RFID networks are mostly cross-organization. As a result, it is not only technically difficult to utilize centralized models, but also infeasible in the sense of business development.

In this chapter, we present our proposed distributed traceability model [SWR10, WSR11b, WSR11a]. This model is inspired by the distributed model introduced in IBM Theseos [CKS07b] (Section 2.4.3). However, our model is more sophisticated because it explicitly defines the traceability data structures, including movement, path and related measurements. Based on this model, we propose a Peer-to-Peer architecture for traceable RFID networks. This architecture sup-

ports all kinds of queries efficiently and is highly scalable and flexible. It is built on the top of DHT (**D**istributed **H**ash **T**able) overlay network. For quickly traceability query processing, we index an object at a node determined solely by the object's ID. Unlike Theseos, our architecture does not require synchronization with enterprise data. Contrarily, we propose an algorithm to maintain the model in a P2P fashion.

In order to further reduce the bandwidth costs and balance the workload among all the nodes, we introduce an enhanced algorithm to maintain the traceability model. We extract the key factor which influences both the bandwidth cost and load balancing, and propose the formula to determine its value in different environments (i.e., bandwidth first or load balancing first). In addition, a set of replication policies is proposed to provide robust, highly available service.

This chapter is organized as follows. In Section 3.1, we introduce the traceability model MOODS (**M**odel for m**O**ving **Object** in **D**iscrete **S**pace). In Section 3.2, we introduce a Peer-to-Peer traceable RFID network architecture built on the top of MOODS and the algorithm used to maintain the traceability data structures. In Section 3.3, we propose an enhanced algorithm to maintain the model, and the formulas to decide the key factor. We then introduce traceability query processing algorithms in the architecture in Section 3.4 and the fault recovery policies in Section 3.5. Finally, we discuss some related work in Section 3.6 and conclude this chapter in Section 3.7.

3.1 The MOODS Model

In existing models for moving objects, both time and space domains are continuous. This is necessary because the query result domain is also continuous. However, in many traceability applications (e.g., RFID-enabled supply chains), objects are only captured at fixed locations (signified by *nodes* and *receptors*). The queries refer to a finite set of fixed locations instead of a point or region in an infinite space. The traditional continuous models can be adopted in these applications. However, they are unnecessarily heavy because of the definition and maintenance of various spatial elements. We therefore propose a new model that can represent moving objects and their attributes in an economic way, namely *MOODS* (a Model for mOving Objects in Discrete Space).

3.1.1 The Key Traceability Functions

In this section, we first revisit the notations in Section 2.2.1 and use them to define the basic operations that should be provided by a traceability model.

$$\mathcal{L}(o,t) : \mathcal{O} \times \mathcal{T} \mapsto \mathcal{V} \tag{3.1}$$

Equation 3.1 shows the signature of the function for locating an object. Given an object o in the object set $\mathcal{O}$ and a point of time t in the time domain $\mathcal{T}$, the locating function $\mathcal{L}$ derives the location where the object o was/is/will be at t. If o is not in the system, the result of $\mathcal{L}$ is *nil* (indicating "nowhere").

In RFID traceable networks, the time domain $\mathcal{T}$ is continuous because readers are working continuously to capture moving objects. The time-parameterized queries may take any moment as an input. However, as we discussed in Section 2.2.1, the space domain is discrete. Instead of a continuous infinite domain,

we define the space domain as a finite set of *nodes* $\mathcal{V} = \{v_1, v_2, ..., v_m\}$ (Same as the one defined in Section 2.1.2). The set is dynamic since new nodes can join and existing nodes may leave the network. The object domain is the same as the existing models, i.e. $\mathcal{O} = \{o_1, o_2, ..., o_n\}$.

The function $\mathcal{L}$ essentially locates an object. Another important function is to find the trace of an object. Similar to $\mathcal{L}$, we define the trace function $\mathcal{TR}$ as:

$$\mathcal{TR}(o, t_{start}, t_{end}) : \mathcal{O} \times \mathcal{T} \times \mathcal{T} \mapsto \mathcal{P} \tag{3.2}$$

$$\mathcal{P} : \{(v_{k_1}, v_{k_2}, \ldots, v_{k_p}, \ldots, v_{k_l}) | v_{k_p} \in \mathcal{V}, 0 < l \leq m\} \tag{3.3}$$

Given an object o and a time range (t_{start} to t_{end}), $\mathcal{TR}$ finds the trace of o during that time frame. $\mathcal{P}$ denotes the domain of *path*. A path is a sorted list of nodes (can be empty) in $\mathcal{V}$ as defined in Section 2.2.1. The sorting is done by the order of the nodes visited by object o (i.e., by time).

3.1.2 The Design of the MOODS Model

The basic idea of MOODS is that instead to model the RFID events (like the DRER model 2.3.1), we try to describe the change of locations, which is the base for traceability. The schema of MOODS is very simple (a single relation):

$$Stay : \{\underline{ID, start\_time}, from, end\_time, to\} \tag{3.4}$$

This *Stay* relation represents two separate events. The `from` and `start_time` attributes represent the source and the time of arrival, meanwhile the `to` and `end_time` attributes represent the destination and the time of departure. It

should be noted that `to` and `end_time` can be *nil* because objects can stay at a node indefinitely. The primary key of *Stay* relation is {ID, start_time}, because an object can visit a node for more than once. We do not use {ID, from} as primary key because an object can visit a node from the same source for more than once.

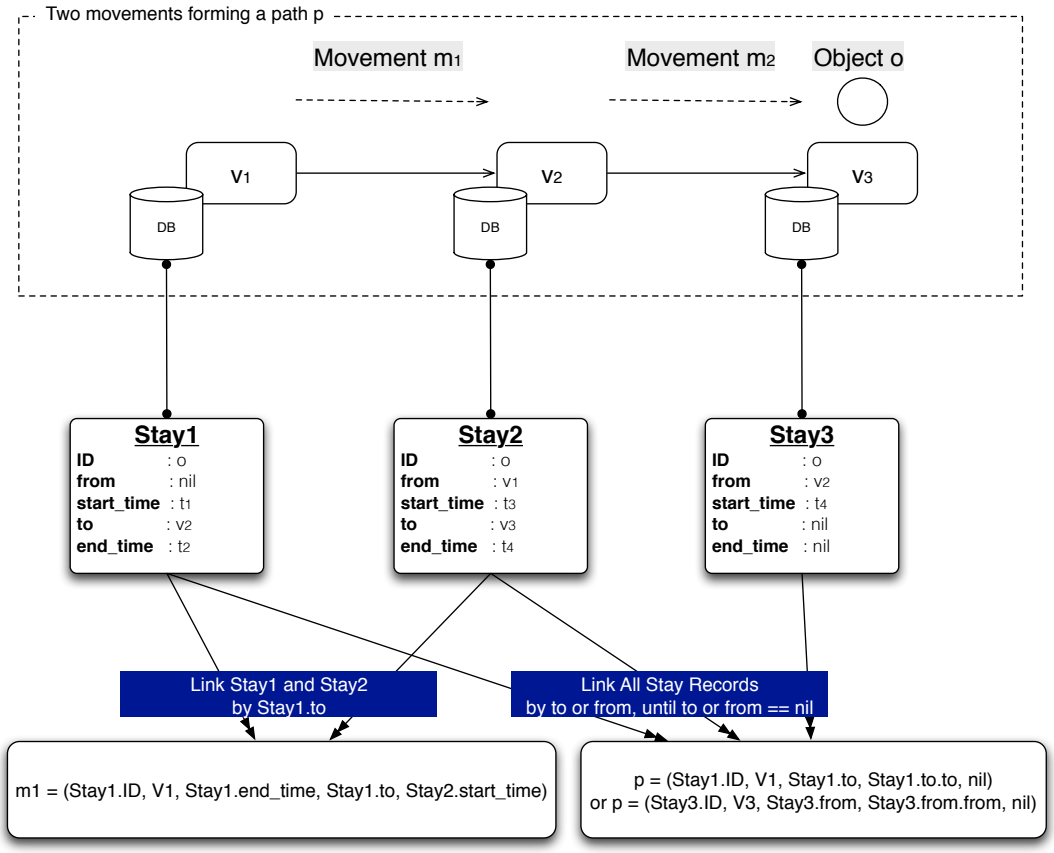


Figure 3.1: MOODS Model Design Overview

Using the *Stay* relation, we can represent the distributed traceability structures easily. Figure 3.1 illustrates the *Movement* and *Path* structure in a distributed environment with *Stay*. *Movement* can be represented by linking two *Stay* record. This can be done at both nodes (source or destination, only the representation from source is shown in the figure), yielding two different represen-

tations (for convenience of presentation, when we use "source" or "destination" in the equations, they are presenting the *Stay* record.):

$$m = (ID, source, source.end\_time, source.to, source.to.start\_time) \tag{3.5}$$

$$\begin{aligned} m = (&ID, destination.from, destination.from.end\_time, \\ &destination, destination.start\_time) \end{aligned} \tag{3.6}$$

The two forms are semantically same, however they are used at different nodes. It is clear that at the source, Equation 3.5 should be used and Equation 3.6 should be used at the destination.

Similarly, path can also be composed in two ways:

$$p = (ID, source, source.to, ..., until\ to\ is\ nil) \tag{3.7}$$

$$p = (ID, destination, destination.from, ..., until\ \ from\ is\ nil)\ .reverse \tag{3.8}$$

Equation 3.7 is used at the source and Equation 3.8 is used at the destination.

Although *MOODS* is defined very simple, it is powerful. Using the equations above, we can answer the two key functions introduced in Section 3.1.1. Given an object o, suppose the queries are issued from a node v[1] which is on o's moving path, we have (please note that the *dot* operation may be a remote call to another node):

$$\begin{aligned} \mathcal{TR}_v\,(o, t_{start}, t_{end}) = (&from, from.from, ..., until\ \ from\ is\ nil\ or\ start\_time < t_{start}) \\ &+v + (to, to.to, ..., until\ to\ is\ nil\ or\ end\_time < t_{end})) \end{aligned} \tag{3.9}$$

[1] $\mathcal{L}$ and $\mathcal{TR}$ are subscripted with v to indicate that v is the node where the queries are issued

$$
\begin{aligned}
&if\ t\ <\ start\_time\ :\\
&\quad \mathcal{L}_v(o,t) = from\ or\ from.from\ or\ ..., where\ from\ is\ nil\ or\ start\_time < t\\
&if\ t\ >=\ start\_time\ and\ t\ <= end\_time\ :\\
&\quad \mathcal{L}_v(o,t) = v\\
&if\ t\ >\ end\_time\ :\\
&\quad \mathcal{L}_v(o,t) = to\ or\ to.to\ or\ ..., where\ to\ is\ nil\ or\ end\_time > t
\end{aligned}
\tag{3.10}
$$

We can see that the *MOODS* model can answer the two queries in a distributed way just by the `from` and `to` attributes. Unfortunately, as we discussed in Chapter 2, these two attributes are not available unless synchronizing the model with other enterprise data. Also, the above two equations have a strong assumption that *the queries are issued from a node v which is on o's moving path.* Although, in most cases this is true, we cannot accept this assumption in all the applications. In the next section, we introduce a P2P architecture working on the top of the model, which can not only get the two attributes but also use them to answer kinds of queries issued from any node in the network.

3.2 A P2P Traceable RFID Network Architecture

Our architecture is built on the top of **D**istributed **H**ash **T**able [BKK03] (DHT). The basic idea is to index ($DHT.put(key, value)$) an object in the DHT overlay network. In our design, there are two sets of DHTs. For the first one, the key in the DHT is the ID of an object, and the value is its latest location. For the second one, the key is the ID of a node, and the value is its URI. We name the two DHTs DHT_o and DHT_v.

Figure 3.2 illustrates the workflow when a movement happens.

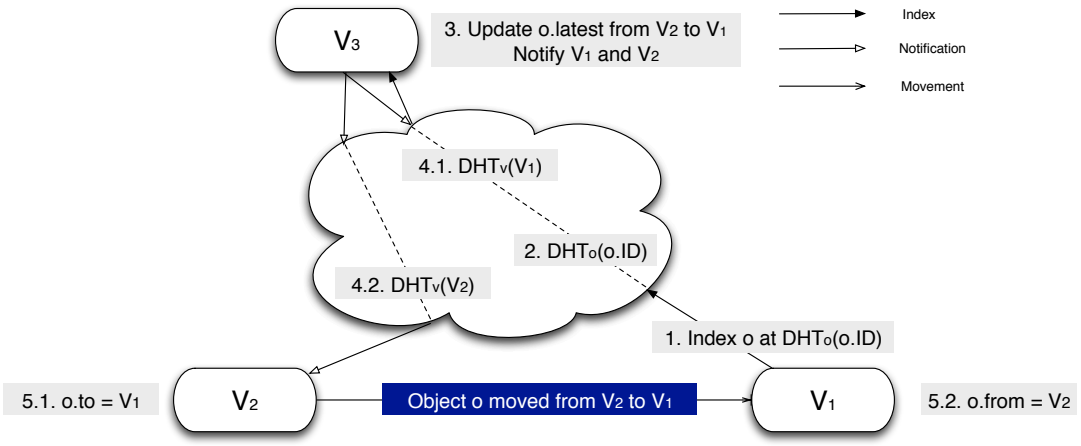


Figure 3.2: P2P Traceable RFID Network Workflow

When object o moved from node v_2 to node v_1, MOODS creates a new record (o, nil, nil, v_1, t_1), then an indexing message (message 1) is sent to the node $DHT_o(o.ID)$. Essentially, this message is a DHT put operation, i.e., the latest location (v_1) of o is **put** into the DHT and stored at $DHT_o(o.ID)$, which is v_3 in this example. v_3, which we call the *Gateway Node* for object o, is responsible to record the latest state of o. After update its record for o by changing $o.latest$ from v_2[2] to v_1, v_3 notifies both v_1 and v_2, about the previous or current states of o, respectively (4.1 and 4.2). Upon receiving of the notifications, v_1 updates $o.from$ as v_2 and v_2 updates $o.to$ as v_1. In this way, the MOODS model is completed. With this structure, we can answer item-level traceability queries by querying the gateway node first to get the address of the node which has the latest information of the given object, then following the linked list of the object's moving path to get the information needed.

The place where the index is stored (*Gateway Node*) is determined solely by the id of the object thanks to DHT's determinism, so that from anywhere in the

[2]The information about v_2 at v_3 was acquired in the same way

network, the object can be located by its ID[3]. The abstract data structure is essentially a double linked list distributed in the DHT network. The head of the list is the *Gateway Node*. When the object moves to a new node, the gateway node is updated and the list is expanded.

In our approach, the information of objects' moving path is stored at nodes in a distributed way as a linked list. An alternative way is to store it only at gateway nodes. However, by doing this, we will not be able to answer range queries (e.g., "how many objects moved from n_1 to n_2 during last month?") because objects traveling together may not be indexed at the same gateway node, due to the uniformity of the hash function. Storing the links at gateway nodes is therefore not a good idea in the sense that we have to flood the network to answer range queries.

We do not require the nodes along the routing path from the original node to the gateway node to store the index to the latest location of an object, although doing so increases the availability of indexes and statistically decreases the time to route queries or indexes to gateway nodes. However, our approach can still achieve similar advantages without this extra storage cost. The reason is that links information is stored along the moving path of objects. Whenever a query is routed to a node that keeps part of the moving path of the corresponding object, the processing of the query can commence without routing to the gateway node.

It is worth mentioning that our architecture is built on top of the existing DHT networks and we do not explicitly deal with the dynamicity of the underlying networks. One reason is that we choose Chord as DHT which adapts well with the dynamic networks. When new peer joins, only a small portion of nodes will migrate their data. Similarly, when a peer leaves, it will migrate its data to

[3]In order to take this advantage, we hash the object's raw id using the SHA-1 function. Thus the ids of the objects and the nodes are in the same space.

another peer [Sto01]. The data migration can be expensive when its amount is huge. However, since the nodes in the traceable networks are reliable servers, they have their own replication policies and the offline time is normally very short. We will discuss more about replication and fault recovery in Section 3.5.

3.3 An Enhanced Model Maintenance Algorithm

3.3.1 The Overview of the Design

A problem of this naive MOODS maintenance method is the *indexing cost*. Whenever an object arrives at a node, the system has to issue 3 messages (1 message from the node and 2 messages from the object's corresponding gateway node) in order to establish the links. This is clearly an expensive approach for large-scale applications where order of magnitude is often thousand. In order to solve this problem while still having the MOODS links established, we next introduce a enhanced maintenance algorithm for MOODS.

In a large traceable network, the data volume might be very high. Indexing each individual object will cause enormous number of messages to flood the network. An obvious solution to solve this problem is to classify the objects arriving at a node within a small period of time into different groups. In particular, objects' raw IDs can be hashed using the secure hash algorithm SHA-1 and grouped by the prefixes of their hashed IDs. Figure 3.3 illustrates how data are grouped and how the gateway nodes are chosen. In general, we define a global prefix length (4 in this example) and group the objects according the prefix of their IDs. The objects are indexed in groups to the gateway node which is chosen according to the prefixes by the DHT.

The index/notify process is the same with the one in Figure 3.2. The only

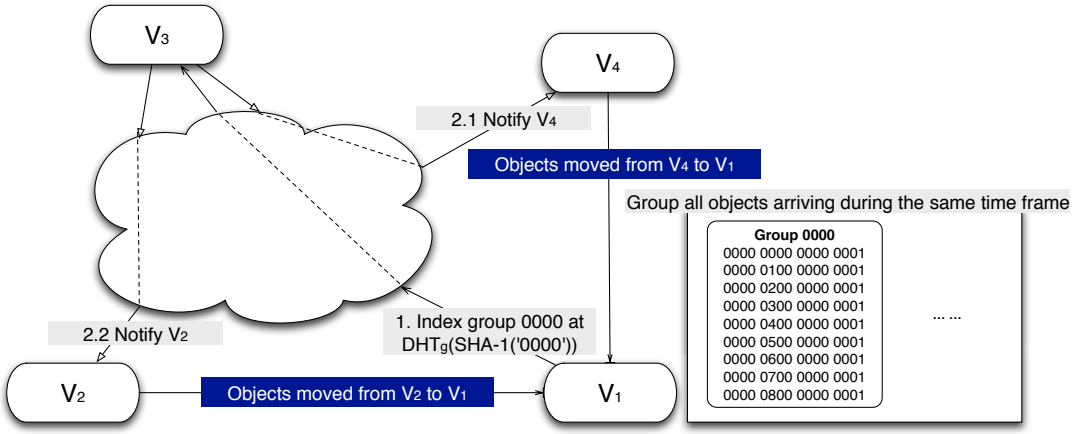


Figure 3.3: Group-based P2P Traceable RFID Network Workflow

difference is that in this approach, the objects are grouped and we index the groups in the DHT, instead of each individual object. Firstly, we define the *Sliding Window*, which is a time frame. During each window, we collect the data and group them according their hashed-ID's prefix. The objects may come from different sources. For example, in Figure 3.3, we have two sources for node v_1: v_2 and v_4 during the same window. At the end of the window, the groups are indexed at the gateway nodes for each group, by a DHT lookup DHT_g. The ID of the group is *SHA-1(prefix)*. The notifications to the sources are in group too.

This approach can significantly reduce the bandwidth cost. For example, if 1024 objects arrived at a node n and we choose a prefix length of 4, there are at most 2^4=16 prefixes. As a consequence, instead of indexing all these 1024 objects, we simply classify them into at most 16 groups by prefixes (with prefixes as the group IDs), and index the groups. The indexing message contains all the objects in the group, and the gateway node still record the latest state for each individual object. Since it is possible to compress the message now, we can save the bandwidth cost significantly.

Based on this concept, we show our enhanced algorithm as pseudo code in Listing 3.1.

Listing 3.1: Pseudo Code : Group-based Indexing Algorithm

```
// The following code is ran at each node as the main event loop
while (true) {
  for each window
    determine the length of prefixes;
    group the collected data by their prefixes;
    send the groups to their gateway nodes;
  end for
}

// The following code is triggered when the node
// receives notifications from gateway nodes
if (message is about source)
  update source for each object in the group;
end if
if (message is about destination)
  update destination for each object in the group;
end if
```

3.3.2 Determining the Width of the Sliding Window

A *group* function is invoked periodically at time intervals of $T_{interval}$ or triggered by other conditions (to be discussed later) to divide the objects captured during this time frame into groups. Two objects belong to the same group when their ids have l prefix bits in common.

$T_{interval}$, is the width of windows of the object stream. We take the objects in the same window for grouping and indexing at one cycle. This parameter is

introduced to restrict the size of the indexing message. Unfortunately, a fixed value of $T_{interval}$ will cause problems when the object stream is unstable. For example, if an object stream suddenly has an extremely high volume (e.g., more products enter the warehouse in one cycle), the number of objects in the same window will be very high thus the size of indexing messages becomes larger.

In our work, we exploit an *adaptive* scheme to determine the width of windows. In particular, we define a maximum time interval, T_{max}, and the maximum number of objects received at each cycle, N_{max}. After T_{max} has passed, or when the number of objects received has exceeded N_{max}, the current cycle ends and the grouping process is started on the received objects during this cycle. Meanwhile, a new cycle is started. In order to limit the message size sent for indexing the objects, we believe an upper bound on the number of received objects in each cycle ($\mathcal{N}_{max}$) is necessary. With a small T_{max}, it is possible to enable real time indexing so that the objects will not be delayed too long for indexing when volumes become very low. The value of T_{max} can be determined by the system's timeliness restriction and is configurable. Similarly, the value of N_{max} can be configured. Also, these values are not necessarily the same universally. Each node can define its own configuration values for them for the best of the performance. Listing 3.2 shows the pseudo code of the algorithm.

Listing 3.2: Pseudo Code : Determining the Width of the Sliding Window

```
while (true) {    // the main event loop
  start := current system time;
  time := start;
  number := 0;
  while ((time - start) < T_max or number < N_max) {
    sleep(1);   // sleep for one second to capture RFID data
    do the indexing;
```

```
      time ++;
      number += number of objects received during last seconds
    }
  }
```

3.3.3 The Key Factor : The Length of Prefixes

The length of the prefix l is the key to determine the total number of groups in the system and should be chosen wisely. If it is too big, the number of groups is high and in the worst case, it is close to the number of objects. As a result, the number of messages will not be significantly decreased. On the other hand, if it is too small, only a small portion of the nodes in the network will be responsible for indexing, thus the work load is not well balanced. Essentially, it is important to find an *optimal* value of l that can guarantee that *almost* every node in the network has the opportunity to index at least one prefix (i.e., group). We denote the probability that a node has at least one group to index as Pr_g. Because prefixes are distributed uniformly at random over the nodes by the hash function, according to probability theory, we have:

$$Pr_g = 1 - \left(\frac{|\mathcal{V}| - 1}{|\mathcal{V}|}\right)^{2^l} \tag{3.11}$$

The optimal value of l should yield the value of Pr_g as close as possible to 1. If $2^l \leftarrow |\mathcal{V}|$, we have:

$$\begin{aligned}\lim_{|\mathcal{V}|\to\infty} Pr_g &= \lim_{|\mathcal{V}|\to\infty}\left(1-\left(\frac{|\mathcal{V}|-1}{|\mathcal{V}|}\right)^{|\mathcal{V}|}\right)\\ &= 1-\lim_{|\mathcal{V}|\to\infty}\frac{1}{\left(1+\frac{1}{-|\mathcal{V}|}\right)^{(-|\mathcal{V}|)}}\\ &= 1-\frac{1}{e}=0.6321<1\end{aligned} \tag{3.12}$$

Obviously, when l is $\log_2|\mathcal{V}|$, the probability that all $|\mathcal{V}|$ nodes are used for indexing is only 0.6321, which is not large enough for good load balancing.

When 2^l is $|\mathcal{V}| * \log_2|\mathcal{V}|$, we have:

$$\begin{aligned}\lim_{|\mathcal{V}|\to\infty} Pr_g &= \lim_{|\mathcal{V}|\to\infty}\left(1-\left(\frac{|\mathcal{V}|-1}{|\mathcal{V}|}\right)^{|\mathcal{V}|\log_2|\mathcal{V}|}\right)\\ &= 1-\lim_{|\mathcal{V}|\to\infty}\left(\left(\frac{|\mathcal{V}|-1}{|\mathcal{V}|}\right)^{|\mathcal{V}|}\right)^{\log_2|\mathcal{N}_n|}\end{aligned} \tag{3.13}$$

It is easy to prove that the limit value of Equation 3.13 is 1. Thus we can determine the $\mathcal{L}_p$ value:

$$l \geqslant \log_2(|\mathcal{V}|\log_2|\mathcal{V}|) = \log_2|\mathcal{V}| + \log_2\log_2|\mathcal{V}| \tag{3.14}$$

According to Equation 3.14, as long as l is chosen larger than $\log_2|\mathcal{V}| + \log_2\log_2|\mathcal{V}|$, we have a high probability to give each node in the network a chance to be responsible for a group. In our system, we choose $\lceil\log_2|\mathcal{V}| + \log_2\log_2|\mathcal{V}|\rceil$ for l.

As new nodes join and existing nodes leave, $|\mathcal{V}|$ is *dynamic*. As a result, there

is no precise way to calculate this value. However, there are existing algorithms available to estimate the value of $|\mathcal{V}|$. Interested readers are referred to [JM04] for details. According to Equation 3.14, l increases much slower than $|\mathcal{V}|$. Therefore there is no need to recalculate l every time the network changes. Instead, l can be adjusted at a relatively long interval.

During the bootstrap of the network, however, $|\mathcal{V}|$ should be re-calculated more frequently to quickly detect a change of l because $|\mathcal{V}|$ starts from a small value and changes frequently. According to Equation 3.14, $|\mathcal{V}|$ will also begin with small values and keep changing, thus at the beginning, the number of groups is small and objects are indexed nearly individually. To solve this problem, we introduce l_{min}, a minimum value of l. The value of l is also configurable and applies to all nodes in the network.

3.3.4 Prefix Triangle

After the groups are built, the node that captured objects will send indexing messages to the gateway node for each group. The gateway node is determined by the hash value of the group id. For example, objects belonging to the group "00" will be indexed in the node hash("00")[4]. The indexing message contains the list of objects in the group "00". The gateway node stores the indexing information for each object individually. We apply gzip compression on the message to reduce the size of the message.

However, we cannot simply store and lookup the index at the gateway nodes. There are two issues we need to address because of the introduction of the grouping scheme.

[4]Note that the parameter for the hash function is a string, instead of an integer, because "00" and "000" are different prefixes

Firstly, changes of l cause grouping inconsistencies. For example, at node v_1, l was 2 and we grouped objects 0000 and 0001 into the same group 00. The index was stored at the gateway node hash("00"). Before these objects arrived at node v_2, new nodes joined that caused l to become 3. After objects 0000 and 0001 arrived at node v_2, they are grouped into group 000 and the corresponding gateway node becomes hash("000"). At this moment, the node hash("000") does not have previous information about these two objects and it may assume that node v_2 is the first node on the path of these objects, which in fact should be v_1. A similar situation also happens when the size of the network decreases. This causes the links to be updated incorrectly.

Secondly, when new nodes join the network, the increase of number of nodes does not cause l to increase, there are always at least $|\mathcal{V}| - 2^l$ nodes idle. The load is then not well balanced.

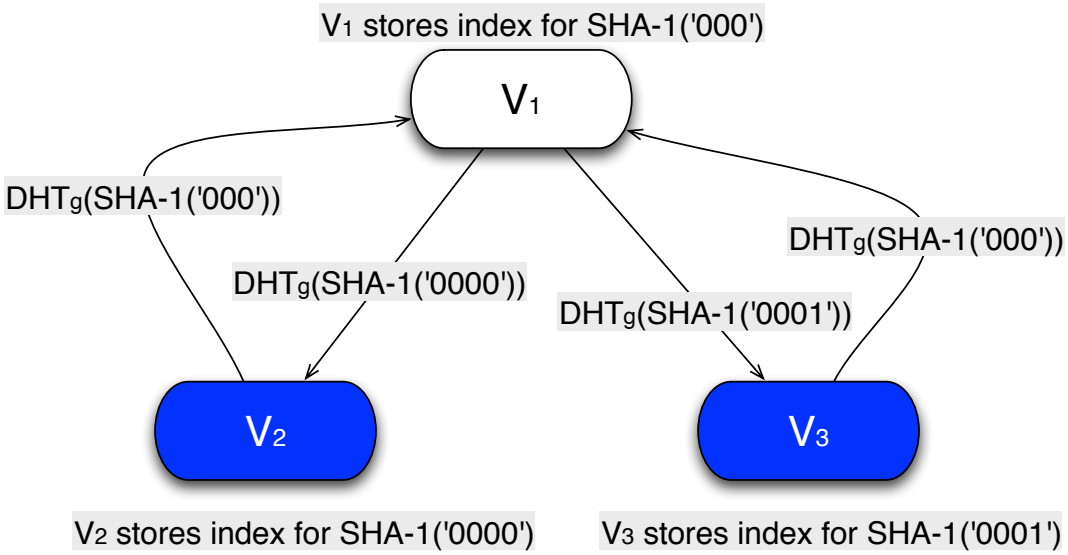


Figure 3.4: Prefix Triangle Example

To solve the above two problems caused by l, we introduce a distributed data structure called *Prefix Triangle*. A prefix triangle consists of three nodes in the

network. The roles of them are not symmetric. Figure 3.4 shows an example of the triangle. Essentially, prefix triangle is a simplified version of Distributed Prefix Hash Tree [RRH04].

The gateway node is responsible for indexing objects whose IDs are prefixed by "000" (current prefix length is 3). The two child nodes are responsible for indexing "0000" and "0001". But the child nodes do not index groups directly, instead they are *secondary storage* for their parent. The parent delegates part of the indexing data according to the *next bit* in an object id after the prefix to the child nodes respectively. With this data structure, even if the network changes so significantly that the value of l changes, we can still find the previous index by following the links between the parent and children in the triangle.

We can also prove that the load balancing is well maintained even when the network constantly changes.

Suppose to increase the current $|\mathcal{V}|$ by 1, we need Δ new nodes to join. We have:

$$\lceil \log_2((|\mathcal{V}| + \Delta) \log_2 (|\mathcal{V}| + \Delta))\rceil - \lceil \log_2 (|\mathcal{V}| log_2 |\mathcal{V}|)\rceil = 1 \tag{3.15}$$

Removing the ceilings, we have:

$$\log_2((|\mathcal{V}| + \Delta) \log_2 (|\mathcal{V}| + \Delta)) - \log_2 (|\mathcal{V}| \log_2 |\mathcal{V}|) < 2 \tag{3.16}$$

It is easy to verify that Δ is less than $3|\mathcal{V}|$. When $\Delta = |\mathcal{V}|$ or $\Delta = 2|\mathcal{V}|$, the inequality holds, while when $\Delta = 3|\mathcal{V}|$, it does not. Since the equation on the

left is a monotonically increasing function of Δ, we can conclude that the value of Δ satisfying the equality in Equation 3.15 must be between $2|\mathcal{V}|$ and $3|\mathcal{V}|$.

In the worst case, there are Δ nodes idle. For a traceable network whose prefix length is l, we use extra 2^{l+1} logical nodes (the child nodes in the prefix triangle) for indexing. The $2^{l+1} + 2^l = 3|\mathcal{V}| \log_2 |\mathcal{V}|$ logical nodes are distributed among at most $\Delta + |\mathcal{V}| < 4|\mathcal{V}|$ physical nodes. By similar inference to Equation 3.14, there is a high probability that all physical nodes have at least one group to index.

With the changes to the network (nodes join and leave), the child nodes may have their children. However, the tree is not necessary. From the analysis above, we can see that the prefix triangle is good enough to maintain a well balanced workload. Maintaining a tree introduces longer delay for looking up. Trees also cost more for processing indexes. For all new objects, unless we fully traverse the tree, it is impossible to determine whether they are new or have historical information stored somewhere in the tree. In this case, each object will be looked up in each level of the tree, so for $|\mathcal{O}|$ objects and a tree of height h, the total number of DHT lookup operations is $h * |\mathcal{O}|$, which clearly is expensive.

Fortunately, we do not maintain the whole tree all the time. Instead, once the network size changes and causes the prefix length to change, we start a *splitting-merging* process. If the prefix length increases, then the child nodes in the triangle become parent nodes in new triangles. The data stored in the old parent will all be delegated into the two new parent nodes which are its child nodes. Similarly, if the prefix length decreases, the parent node in the triangle becomes a child node of another node that is indexing a shorter prefix, the parent node's two child nodes migrate the data they are indexing to the parent node. Thus eventually we always maintain only triangles, instead of trees. To look up an object which does not exist locally, we only need to ask the parent and its two children. Taking

Figure 3.4 as an example, after the prefix length increases from 3 to 4, the node indexing prefix "000" will split all its data into "0000" and "0001" according the fourth bit of their IDs. It is not necessary to start the splitting-merging process as soon as the prefix length changes. Instead it can be done during the system's free time.

3.3.5 The Group-based Indexing Algorithm on Prefix Triangle

Figure 3.5 summarizes our indexing algorithm. The *index* algorithm first tries to update the index stored locally (line 1) and gets a list of objects whose index is stored either in ascents or descents by a set difference operation (line 2). Then we refresh the local indexing records by searching up and down in the tree (lines 4 and 5) for the objects in the difference set. After the index of all objects are downloaded to the local storage, the index will be grouped by source nodes and sent back to child nodes (not shown in the algorithm).

The two refresh procedures are used to retrieve the indexes by traversing the tree up and down, respectively. Before sending the fetching request to the respective nodes, the object list is filtered by the prefix (the filter function in line 4 of $refresh\_from\_ascent$ and lines 1, 6 of $refresh\_from\_descent$) for pruning. The address of the parent and children can be cached to save the cost of DHT lookup.

For the delegate procedure, the system first finds the objects not having been stored (line 1). Then it checks whether it is necessary to delegate some records to the two child nodes (line 2). There can be different strategies to determine this. For example, whether the local storage for this prefix exceeds a certain amount, or whether there have been a number of records older than a pre-configured time. If a delegation is required, we select the earliest $\alpha * |\mathcal{O}|$ $(0 < \alpha \leq 1)$ objects indexed

Algorithm : Index

Input: prefix pf with length l and objects $\mathcal{O}$ belong to the group $hash(pf)$
Output: None (the algorithm only updates indexing information)

1: update the index for objects $local.objects \cap \mathcal{O}$
2: $\mathcal{O}' \leftarrow local.objects - \mathcal{O}$ // get the set of objects which are not stored locally
3: **if** $\mathcal{O}'$ is not empty
4: $refresh\_from\_ascent(\mathcal{O}', p)$
5: $refresh\_from\_descent(\mathcal{O}', p)$
6: **end if**
7: $delegate(objects)$

refresh_from_ascent

Input: prefix pf with length l and objects $\mathcal{O}$ belong to the group $hash(pf)$
Output: None (the algorithm refresh local index with ascents)

1: $i \leftarrow 1$
2: **do**
3: $pf' \leftarrow pf.substring(1, l - i)$
4: $\mathcal{O}' \leftarrow filter(\mathcal{O}, pf')$
5: **if** $\mathcal{O}' = \Phi$ **break**
6: update index with information from gateway node for pf'
the gateway node for pf' also updates its index
7: $i \leftarrow i + 1$
8: **while** there exists gateway node for prefix pf' and $i \leq l - l_{min}$

refresh_from_descent

Input: prefix pf with length l and objects $\mathcal{O}$ belong to the group $hash(pf)$
Output: None (this algorithm refresh local index with descents)

1: $\mathcal{O}' \leftarrow filter(\mathcal{O}, pf$ + '0')
2: **if** $\mathcal{O}'$ is not empty
3: update index with information from gateway node for pf + '0'
the gateway node for pf + '0' also updates its index
4: $refresh\_from\_descent(\mathcal{O}', p$ + '0')
5: **end if**
6: $\mathcal{O}' \leftarrow filter(\mathcal{O}, p$ + '1')
7: **if** $\mathcal{O}'$ is not empty
8: update index with information from gateway node for pf + '1'
the gateway node for pf + '1' also updates its index
9: $refresh\_from\_descent(\mathcal{O}', p$ + '1')
10: **end if**

delegate

Input: prefix pf with length l and objects $\mathcal{O}$ belong to the group $hash(pf)$
Output: None (the algorithm only updates indexing information)

1: $\mathcal{O}' \leftarrow$ objects not stored in any node in the tree
2: **if** delegation is required
3: select $\alpha * \mathcal{O}'.count$ earliest local indexing records as $\mathcal{O}''$
4: delegate $\mathcal{O}''$ to the two children according to the l bit of their IDs
5: **end if**

Figure 3.5: Algorithm for Indexing a Group of Objects

at this gateway and delegate them to the two child nodes. The delegation is similar to the *FIFO* cache replacement policy. This is based on the observation that the latest records are more likely to be read and updated in the near future. Here α is a global configuration to control the number of objects to be delegated. It should be noted that the splitting-merging process is trivial and we have not included the details of the algorithm.

In Figure 3.5, we always try to traverse a tree. This is not only because we do not do immediate splitting-merging, but also because the detection of prefix length changes at different nodes is not synchronized, thus it is possible that at some time (usually after a prefix change), there are trees existing in the system.

3.3.6 Algorithm Analysis

The indexing process consists of three phases: namely *grouping*, *routing* and *index-persisting*. The grouping phase simply scans the $|\mathcal{O}|$ new objects. Its complexity is $\Theta(|\mathcal{O}|)$.

Chord routing takes $O(\log_2 |\mathcal{V}|)$ hops with high probability. To route the objects in 2^l groups will take $O(2^l \log_2 |\mathcal{V}|)$ hops in total. When objects are routed individually, it takes $O(|\mathcal{V}| \log_2 |\mathcal{V}|)$ hops. Since the number of received object $|\mathcal{V}|$ can be very large, while $2^l = |\mathcal{V}| \log_2 |\mathcal{V}|$ is relatively small, our grouping index algorithm can significantly decrease the P2P routing cost.

However, the index-persisting phase is complicated because of the *Prefix Triangle* structure and the possible trees extended from it. In the best scenario, all the indexing information for new objects are stored in the gateway node, then the persisting will only involve local lookup and storage. The complexity is $O(|\mathcal{O}|)$ in the measure of database operations.

In the worst scenario, all the new objects have their history indexes stored

in either ascent nodes or descent nodes, or all the objects are new to the whole network thus there is no historical information in the tree (then the tree has to be fully traversed). Index persisting may involve both refreshing (i.e., $refresh\_from\_ascent$ and $refresh\_from\_descent$) and delegation. Suppose that the tree is complete binary and its height[5] is h, and all the historical indexing information is stored either in the deepest leaves or in the root. Therefore, it requires at most h DHT lookup to obtain the information. So the complexity of the worst case of the index persisting is $O(h * |\mathcal{O}|)$. However, as the tree will be split or merged when prefix length changes, its height is well controlled. In most cases h is 1 (only children) or 2 (parent and children). Then the index persisting phase can be done in constant time.

3.4 Traceability Query Processing Algorithms

In this section, we introduce the algorithms to process two kinds of tracing queries, namely *item level* and *statistical* queries.

3.4.1 Item Level Queries

To perform the trace function $\mathcal{TR}$ (see Section 3.1.1), we need to find the gateway node of the given object. From the gateway node, we can discover the node where the object is seen for the last time and simply traverse back using the MOODS links. In addition, as mentioned in Section 3.2, if any node along the route from the query requesting node to the gateway node has the information of the object, the trace query can be processed from this node by traversing backward and forward using the linked list. In this way, we do not have to find the gateway

[5]The height of a tree is defined as the number of edges from the root to the deepest leaf.

node of the object. With $\mathcal{TR}$ solved, it becomes straightforward to process other kinds of item level queries, include but not limited to:

- Dwell time query. For example, "How long did o_1 stay at node v_1?".
- Arrival time query. For example, "When did o_1 arrive at node v_1?".
- Source node query. For example, "Where did o_1 come from before it arrived at node v_1?".

3.4.2 Statistical Queries

Statistical queries are used to find out the aggregated information for a given criteria. For example, "How many objects moved from node v_1 to node v_2 during last month?". In our architecture, with the help of MOODS, this query can be processed at either node v_1 or v_2, despite the fact that it involves two nodes. Suppose the query is being processed at v_1:

```
SELECT COUNT(*) FROM RECORDS
WHERE TO = v2 AND
END_TIME BETWEEN '2011-04-01' AND '2011-04-30'
```

This query is *segment-oriented*. A more complex range query can be *path-oriented*. For example, "How many objects moved along the path $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k$ during last month?". With our solution, this query can also be efficiently answered. First, the node initiating the query finds the nodes involved in the query via Chord lookup interface. Then it asks the node v_i (i from 2 to k) to answer the query: "Which objects moved from node v_{i-1} during last month?".

This query can be easily answered. Finally, the results from the $k-1$ nodes are intersected to get the result.

Other range queries regarding either path or segment can be handled in a similar way.

3.4.3 Algorithm Analysis

3.4.3.1 Item Level Queries

There are two cases for query processing according to where the $\mathcal{L}$ function (see Section 3.1.1) is answered, which will be discussed separately in the following.

Gateway. If there is no relevant node along the routing path from the node where the query is issued to the gateway node, then the $\mathcal{L}$ is answered by the gateway node. The routing process takes $O(log_2|\mathcal{V}|)$ hops. At the gateway node, there is a high probability that the *lookup* algorithm in takes $O(1)$ hops.

Intermediate Node. If during the routing, a node along the routing path has the information for the queried object, the routing will be terminated and the intermediate node will start to process the query. Although the routing cost remains $O(\log_2|\mathcal{V}|)$, the constant coefficient is reduced.

After the node v where the query can be answered is found, the complexity of query processing is solely based on the type of the query. For example, $\mathcal{L}$ takes $O(l)$ (l is the length of the object's lifetime trajectory) DHT lookups in the worst case (when the object was at the end or start of the trace for the given time) and $O(1)$ for the best case (when the object was at v for the given time). However, for a trace query $\mathcal{TR}$ that requires the lifetime trajectory to be found, it will always take $O(l)$.

3.4.3.2 Statistical Queries

For statistical queries that only involve segments, the cost is solely based on the query processing method of local database. There is no network cost, except when the query is initiated from a third party node (This rarely happens), a Chord lookup is used to first find the node involved which cost $log_2|\mathcal{V}|$ hops.

For statistical queries that involve paths, the length of paths determines the cost. Suppose the length of a path involved in a query is l, $l * log_2|\mathcal{V}|$ hops are used to find all the involved nodes. All the other calculations are done locally in either memory or DBMS, which can be ignored compared to the cost of P2P lookup.

3.5 Replication and Fault Recovery

Due to limitations of the RFID hardware and dynamic network environments, the recorded data do not perfectly map the real movements of objects. Uncertainty is a very broad topic and it is not a single layer problem. For example, research that addresses uncertainties spans from data storage and data modeling to query processing and knowledge engineering [WRS11]. In this section, we briefly discuss the typical uncertainty problems in the traceable RFID networks and how they are handled.

3.5.0.3 Hardware Faults

A reader may miss identifying an object or a temporal malfunction of a device may cause a systematic error in events generated at a node. A missed tag read results in no data collected since data, such as the identifier stored on the tag, is not captured by the reader. Figure 3.6 shows an example of this scenario. When

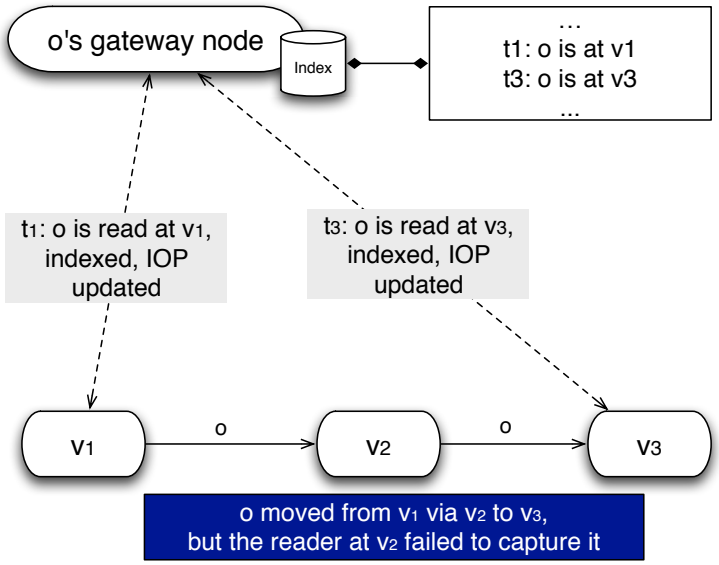


Figure 3.6: An Example of Missing Readings

object o arrives at node v_2, it fails to be read by the reader. Consequently, the record at the corresponding gateway node is outdated. When object o arrives at node $\mathcal{C}$, the record is corrected. A missing reading is a fatal and unrecoverable problem to all RFID systems. In this example, from the point of the system, object o never appears at node v_2. We handle this situation and the outdated record at the gateway node is corrected as soon as the object moves to a new node.

3.5.0.4 Network Faults

A node may leave the network for various reasons, such as network or power outage, system failure etc. When this occurs, all data stored at that node become unavailable. A general approach for data recovery in P2P system is replication. In PeerTrack, we replicate the RFID data and its index in different ways.

The index is replicated by using a secondary hash function when choosing

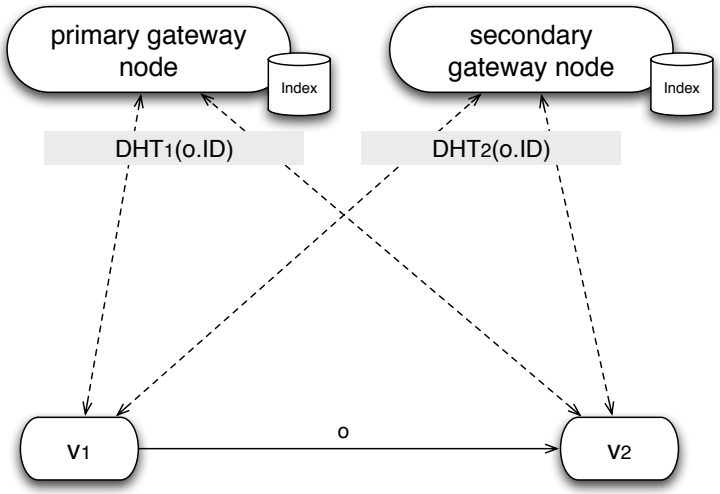


Figure 3.7: Replication of Indices

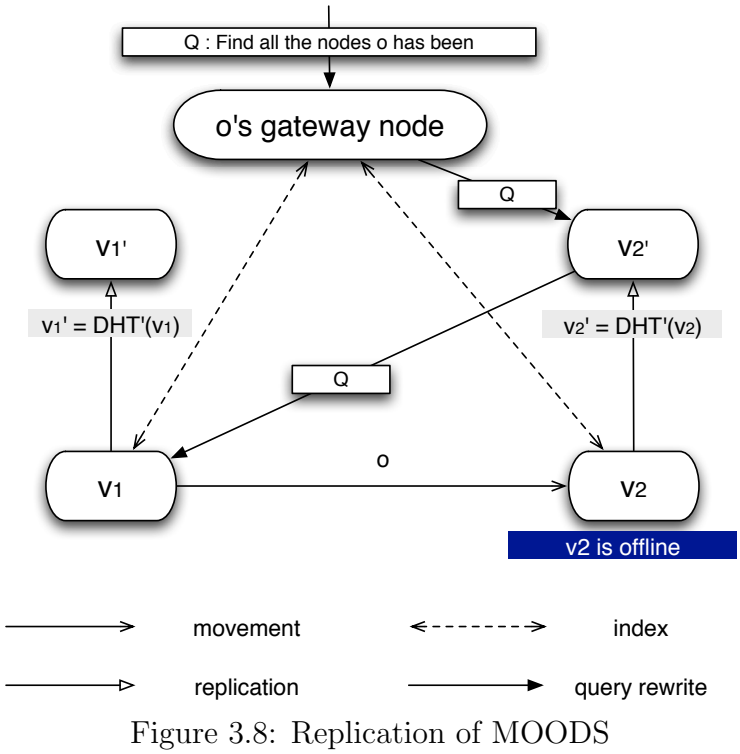


Figure 3.8: Replication of MOODS

gateway node. This is depicted in Figure 3.7. In addition to the primary hash function (DHT_1), a secondary hash function (DHT_2) is used to choose a secondary gateway node. When the primary gateway node leaves the network, the system upgrades the secondary one to primary and use it to maintain the index. The approach is flexible in the sense that if a third replication is necessary, the only change is to add a third hash function.

The MOODS data is replicated at a node chosen by a unified hash function DHT', i.e., the node that keeps the replication of node v_1, v_1', is chosen by $DHT'(v_1)$. Figure 3.8 shows an example of using IOP replications to answer tracing queries. When v_2 is offline, v_2' will be queried. As a result, the routing path of the query becomes $Gateway \rightarrow v_2' \rightarrow v_1$, instead of $Gateway \rightarrow v_2 \rightarrow v_1$.

3.6 Related Work

Successfully tracing objects in a distributed environment is not a single layer problem. In this section, we overview the relevant techniques to our work.

The first step of modeling moving objects is to abstract the basic elements such as time, region and velocity. [SWC97] introduces a data model called MOST (Moving Objects Spatio-Temporal) for databases with dynamic attributes, i.e., attributes that change continuously as a function of time. A language called *Future Temporal Logic* (FTL) has been designed to support queries for dynamic attributes. This work models the moving objects and their attributes in a generic way that can be easily adapted into various applications in different domains. [EGS98] proposes a similar model with a method to represent the continuous attributes such as time and space discretely.

Later works mostly use the same or similar idea. With the elements modeled,

it is possible to answer basic queries such as location of an object at a certain time. In order to answer more complicated queries, such as an aggregate query "how many objects are in region $\mathcal{R}$ now?" and a future state query "where will the object $\mathcal{O}$ be after one hour?", various complex and domain-specific models have been developed. For example, in [SPT04], adaptive multi-dimension histogram (AMH) is used in answering aggregate queries about past, present and future. It can estimate the number of objects that will satisfy some spatial condition for a near future time. It does not require the knowledge of velocity vectors, but uses an *exponential smoothing* based stochastic approach. Since this kind of work focuses on aggregate queries, it does not address the single-instance queries. In [CAA01], the moving objects are associated with four variables (starting time, starting location, destination, initial velocity) in order to predict their future locations. There are some other works focusing on various problems of modeling and querying moving objects by employing different techniques. However, they share some common characteristics. Firstly, space and time are modeled as continuous attributes, although the underlying representation in physical storage can be discrete. Secondly, most of the works define region or similar concepts that cover a finite area in a continuous infinite domain in order to answer range queries.

Index is often used to quickly answer range queries for a specific attribute. For example, if the space is divided into cells and objects are indexed by cell, it is easy to answer queries such as "how many or which objects are in cell c?". However, dividing the space into fixed-size cells does not work well in dynamic environments because the boundary of the space must be decided in advance. Early works, including R-Tree, R*-tree, TR-tree, TB-tree, TPR-tree, TPR* R-tree and some other similar trees [MGA03], dynamically decide which point or region to index and optimize the indexing process in different ways.

[JLO04] considers streaming data that require frequent updates and proposes an efficient B^+-Tree based indexing method which represents the moving-object locations as timestamped vectors. SINA (Scalable INcremental hash-based Algorithm) [MXA04] supports concurrent and continuous spatial-temporal queries by abstracting the continuous queries as a spatial join between a set of moving objects and a set of moving queries. [PCC04] indexes the *predicted trajectories* to quickly predict future locations of moving objects. Many recent works such as [BMN08, GCL08, LDR08, TYJ09, ZCJ09] focus on other specific problems and provide corresponding solutions. These index methods all operate in centralized environments.

In recent years, some generic peer-based database management systems [HRZ08] have been proposed to support large volume data and complicated queries. PIER [HHL03] presents a distributed query engine based on a P2P-based overlay network.However, PIER requires every data tuple to be shared in the network, which brings two significant issues. Firstly, the amount of messages that are sent for indexing are proportional to the number of data tuples. It is problematic when the approach is used in large databases with millions or billions of tuples. Secondly, the data is made public in PIER thus there is no privacy among participants. PISCES [WLO08] identifies a subset of data tuples to index based on some criteria (e.g., query frequency, update frequency, indexing cost). This significantly saves the cost of indexing and storage. However, because its objective is to improve efficiency of range queries, PISCES does not work well with single instance query. If the column in the search criteria that is used to search for an individual object is not indexed, the query has to be flooded into the whole network. Although these generic solutions lays the foundation for peer-based spatial databases, they are not dedicated to efficiently manage spatial data and answer spatial queries. [MS05] presents an analytical model to predict the cost of query

algorithms based on query location, query size and the moving objects' distribution so that the final scheme chosen to perform the query is optimal. [LKG03] proposes a distributed hash technique to answer range queries, which scales well under certain assumption about the query distributions. [ZKC04] introduces a middleware design based on distributed R-tree and Quadtree to support both range and kNN (k Nearest Neighbors) queries.

The work presented in [THS07] extends Quadtree index into a P2P network to answer range queries efficiently while keeping the load on the nodes in the network well balanced. [LZ05] provides a linear yet distributed structure that facilitates multiple search paths to be mixed together by sharing links. These distributed index methods are all based on spatial elements such as point, link or region thus they work well for range queries but not single-instance queries. In [ZWS10], the authors proposed a generic model to deal with the event matching problem of content-based publish/subscribe systems over structured P2P overlays. This model is useful in a distributed RFID system if it is event-driven instead of query-driven.

3.7 Summary

In this chapter, we have presented a distributed traceability data model and a generic approach that enables applications to share traceability data across independent enterprises in a P2P fashion. Our main contributions include:

- We built our approach on the top of a DHT-based overlay network. The partners do not need to send all the data to a centralized database so that they can fully control the access to only allow certain partners.
- Objects are indexed at deterministic gateway nodes that are responsible

for updating objects' status at the source and destination nodes for their movements. In this way, the moving path of objects are established, which are critical to support efficient processing of traceability queries.

- To reduce the indexing overhead from massive volumes of data in large-scale traceability applications, we further proposed an enhanced group-based indexing approach.

We have implemented the proposed model and architecture and conducted an extensive set of experiments for the performance evaluation. We will report the details of the implementation and performance evaluation in Chapter 6.

Chapter 4

Mining Moving Patterns from Distributed RFID Streams

In Chapter 3, we introduced the model MOODS and a P2P architecture to enable traceability in item level. However, in some other scenarios, it is also important to be able to find out the implicit characteristics of the business workflows behind the scene. For example, in a supply chain network, a manager may be interested in how the sales change in terms of time of the year. This is difficult to be answered by MOODS or other **O**n**L**ine **T**ransaction **P**rocessing (OLTP) systems, because this requires to go through all the data.

Some of the existing mining methods, such as regression analysis, require to read the data for multiple times. Meanwhile, others require to prepare the data beforehand, e.g., data cubes. These methods are not suitable for processing RFID data, because through the connection of billions of tags and sensors, applications will generate RFID streams with unprecedented volume. In addition, RFID networks are federated systems formed by autonomous nodes, just like the Internet itself. The sovereignty of data should therefore be maintained (See Chapter 2). Indeed, object movements and related data are valuable business information that companies may be very reluctant to put such data in a shared central warehouse.

Instead of maintaining the exact movements of objects, we propose in this paper a probabilistic model that maintains the *object flow patterns*. An object flow pattern is a function of time, which describes the volume/frequency of object movements at a specific time. Our goal is to extract and model the patterns of object flows from high-volume, highly dynamic RFID data streams in autonomous network environments. With these patterns, the efficiency of distributed data processing and mining can be significantly improved. It should be noted that this is a challenging task because the movement of an object is implicit. It is impossible to acquire such information without querying other nodes in the network. Ideally, the model should be established with a limited number of network calls without flooding the whole network. Our contributions are summarized as follows:

- We propose a new model called *Tilted tIme Series of Histograms* (TISH) that combines two techniques, namely *Histogram* and *Tilted Time Frame* [HK06]. Essentially, TISH is a synopsis of the object flow. It represents the patterns of object flow between two nodes for a long history using limited memory. The pattern for a given period of time is represented by one or more histograms. The model suggests the probability that an object comes from a particular node at a specific time. TISH does not require any kind of indices. It is highly efficient in storage and bandwidth.

- We develop a Peer-to-Peer (P2P) architecture and a set of algorithms to establish and maintain the TISH model. To avoid long delays and extra bandwidth usage caused by network queries, we further develop an algorithm to choose the neighbors which are most possible to have the quested information as the target of query rewriting to avoid unnecessary network traffic. To avoid data migration when the underlying P2P layer changes

(nodes leave or join), we introduce a simple but effective data structure for maintaining network topology. Based on the TISH model and the P2P architecture, our proposed algorithm on the item-level tracking and tracing query processing, statistically keeps the number of network calls to a minimum.

The rest of this chapter is organized as follows. We formally define the problems, and discuss why a centralized solution is not feasible in Section 4.1. In Section 4.2, we introduce the architecture of a distributed RFID system which is built based on our proposed model. In Section 4.3 and Section 4.4, we describe the TISH model, and its related maintenance algorithms. In Section 4.5, we analyze the costs in maintaining the proposed model and performing tracking and tracing using our proposed techniques. The related works are discussed in Section 4.6. Finally, Section 4.7 provides some concluding remarks.

4.1 Problem Definition

We define the object flow pattern as two functions: $\mathcal{F}_{out}(t_a, t_b, v_i, v_j)$ and $\mathcal{F}_{in}(t_a, t_b, v_j, v_i)$, where t_a and t_b define the range of time under consideration, v_i and v_j are two ends of the connection. $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$ define the inbound and outbound flow patterns respectively. They are formally defined as:

$$\mathcal{F}_{out}(t_a, t_b, v_i, v_j) = \frac{|\mathcal{M}(t_a, t_b, v_i, v_j)|}{\sum_{v_k \in \mathcal{D}(n_i)} |\mathcal{M}(t_a, t_b, v_i, v_k)|} \tag{4.1}$$

$$\mathcal{F}_{in}(t_a, t_b, v_j, v_i) = \frac{|\mathcal{M}(t_a, t_b, v_j, v_i)|}{\sum_{v_k \in \mathcal{S}(v_i)} |\mathcal{M}(t_a, t_b, v_k, v_i)|} \tag{4.2}$$

Where $\mathcal{M}(t_a, t_b, v_i, v_j)$ movements from v_i to v_j during time t_a to t_b, $\mathcal{S}(v_i)$

(resp., $\mathcal{D}(v_i)$) is the set of source (resp., destination) nodes of the inbound (resp., outbound) connections of v_i. The two functions describe object flow patterns in both time and space dimensions. In a distributed environment, these object flow patterns can be used as heuristic knowledges in the discovery of source/destination node. For example, we can sort the candidate source nodes for an object in the order of $\mathcal{F}_{in}$ and query them sequentially to save time and bandwidth.

If data from different nodes are stored in a centralized database using the schema above, the definitions themselves are efficient enough. Proper indices can improve the performance. However, there are still some significant performance issues in large scale systems where the number of nodes and objects could be very large. We discuss below why a centralized solution does not scale by considering the characteristics of RFID data management.

- *Frequent Updates.* Data in RFID networks are generated as streams and update becomes a frequent operation comparing to traditional database management systems that are optimized for frequent-read scenarios. With a centralized database (or database clusters), frequent updates to the database not only increase the storage cost, but more importantly, cause high costs in index maintenance. For example, for streaming data, a B^+ tree index could be potentially huge. The space cost for the index itself is very huge too.

- *Row Level Security Requirement.* Business applications often require high security. Even in a federated system, the partners want to control their own data. For example, a supermarket wants to hide the buying information about the same product from one supplier to another, while the suppliers can access the data related to their own products. This requires row level authentication and the authentication overhead for space is high.

- *Archiving.* RFID data, similar to other time series data, is sensitive to time. Recent records are more interesting to the analyzers than the distant ones. For the purpose of storage efficiency, it is often necessary to archive the old data in order to make room for the new data. However, different participants may have different definitions of oldness (i.e., when the data should be archived). As a result, when old records are archived, a range query has to be performed. Furthermore, because the records for different nodes are likely to be stored in different pages, after deletion, the pages will have a lot of fragments. The rearrangement of records in these pages could be very costly and time consuming.

- *Mining Efficiency.* Stream mining techniques such as online aggregation, critical layers or popular path materialization all require a significant amount of memory in a central server.

Our goal is to build a distributed model which can be used as a middleware in a federated system. It does not require any centralized server for coordination, nor full access to data at other partners for its establishment and maintenance. The model is expected to be able to represent the pattern defined above for any time-and-node pair, and to be used to expedite distributed tracing and tracking query processing.

4.2 The Architecture for Distributed Stream Mining

The architecture of the mining middleware at a node is depicted in Figure 4.1. The *TISH Model* is updated repeatedly for every event cycle by the *Model Maintainer*. The model maintainer takes a small random *Sampled Data* out of the large volume of *Preprocessed Data* as input, analyzes the sample by querying the

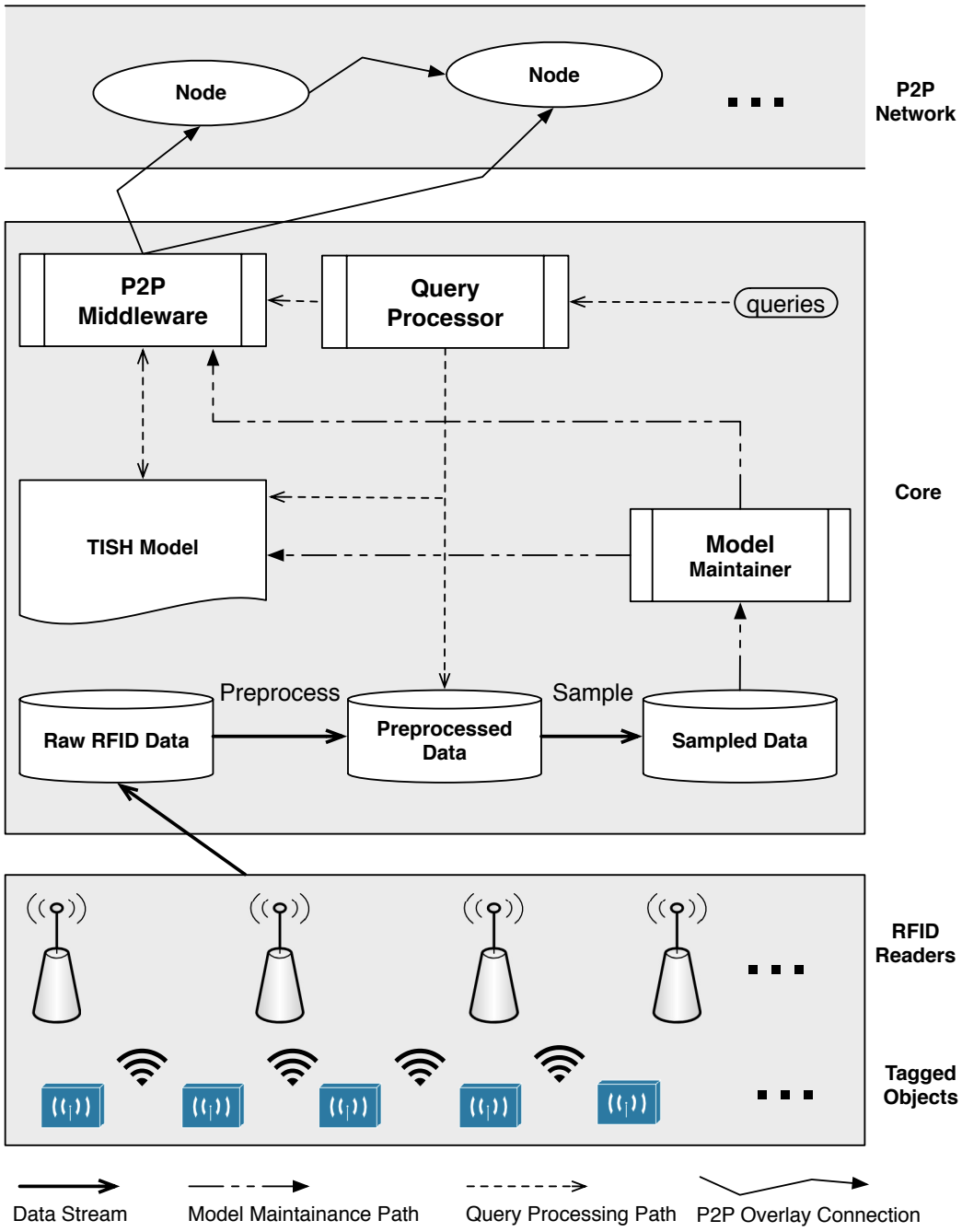


Figure 4.1: The Architecture for Distributed Stream Mining

neighbors via the *P2P Middleware*, and refines the model using the result. The *Query Processor* is responsible for answering queries from either local or remote users. It makes use of the *TISH Model* to find the candidates to rewrite the query if necessary.

We maintain two layers of neighbors for a node v_i in this architecture, namely the *network layer* and the *business layer*. Nodes in the *network layer* are used to maintain the P2P overlay. Since many P2P overlays have been developed (e.g., Gnutella [Rip01], CYCLON [VGS05]), we assume these nodes can communicate with each other through packet relaying and we will not be concerned about how this is implemented in this work.

The *business layer* is the key for object flow pattern abstraction. Neighbors in this layer consist of two sets: the source, and the destination nodes of v_i. For each source (resp., destination) node v_j, we maintain the pattern of object flow (defined in Section 4.1) from (resp., to) v_j to (resp., from) v_i using the *Tilted tIme Series of Histogram* (TISH) model. The details of the TISH model and its maintenance will be described in Section 4.3 and Section 4.4.

This model is built by exploiting the fact that movements of objects are likely to be continuous and bulky in both time and space, so a connection which is active in the previous window is likely to be active in the current window. In other words, an object received in the current window is likely to have originated from one of the source nodes of the active connections in the previous window[1]. Based on this observation, when looking for the source node of an object, it is more efficient to first query the source nodes in previously active connections, which can be obtained by searching the *business layer* neighbors. It is worth mentioning that the unstructured P2P query is much more expensive. We only

[1]The same argument holds for destination nodes.

use it when i) the *business layer* has no nodes; ii) an object is from a node which is not in the *business layer*.

This architecture adapts well with the characteristics discussed in Section 4.1:

- *Frequent Updates.* We sample the input and use only a small portion of incoming data to maintain the model. Thus the system scales well with frequent updates. Also, using the tilted time frame model, we can store a long history of object flow patterns in the main memory.

- *Row Level Security Requirement.* Since data is never stored at central servers, each node can have its own security schema. Each node owns the data physically and fully controls who can access which portion of its data. This model is strictly private, because there is no super user who can access data from every node.

- *Archiving.* Partners can archive their data whenever they deem appropriate, with flexible strategies. The archiving process is fast, because the records are stored in the order of time. To archive the records before a particular date, we only need to find the first record that is younger than the given criteria using a binary search, and move all the pages before it to the archive media.

- *Mining Efficiency.* The model maintained at each node contains the patterns for the object flow. It can be used as a starting point for online aggregation, materialized data cube and data visualization.

4.3 The TISH Model

4.3.1 The Overview of TISH Design

In RFID applications, the object flow among nodes is determined by business actions and follows certain patterns. For example, a supplier sends products to a supermarket on a regular basis, such as once a week or once a month. Different products may (usually) have different patterns, so do the same products from different suppliers. If the patterns are known, we can use them to find out which supplier is most likely to be a source node from which a given object comes.

In some applications, the patterns are not static. Instead, they evolve from time to time. For example, during Christmas, big sales require the supermarkets to order more frequently with more products in each order than usual.

The problem of modeling these patterns can be categorized as *time-series data mining* [HK06]. A popular method is *regression analysis* in modeling time-series data and finding trends. However, we do not use this method because: i) regression analysis cannot cover the time-varying patterns in the sense that it essentially tries to find a function which best fits the given data for all the time; ii) we do not necessarily need a global function in this problem, as we only need local statistics; and iii) regression analysis often requires reading data for multiple times, which is not possible with RFID streams.

Our model is based on two important techniques in data mining: *Tilted Time Frame* and *Histogram*. Tilted time frame is very useful in data stream analysis, because it gives more details on the recent data than on the old data. There have been many ways to design a tilted time frame, but the most important ones are : i) *Natural Tilted Time Frame Model*, ii) *Logarithmic Tilted Time Frame Model*, and iii) *Progressive Logarithmic Tilted Time Frame Model*. We

will not introduce the details of these models here, interested readers are referred to [HK06] and [GHP03].

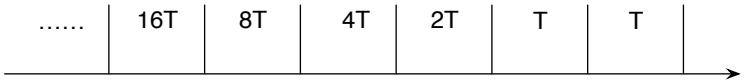


Figure 4.2: The Logarithmic Tilted Time Frame

In this work, we choose **L**ogarithmic **T**ilted **T**ime **F**rame (LTTF) model for its simplicity and flexibility. As illustrated in Figure 4.2, the first (right most) slot represents the data for the time range of T_0, while the i^{th} slot s_i represents a range of $2^i * T_0$. Each time unit (slot) in LTTF occupies the same amount of memory, but the most recent time slot provides the statistics with the finest granularity, while the older ones are with coarser granularities.

We combine the two techniques and propose a new structure, namely *Tilted tIme Series of Histograms* (TISH). By combining the two techniques, it is possible to model the dynamics of RFID streams in both time and spatial dimensions. Tilted time series capture the changes of streams at different times, while histograms measure the distributions of streams from different nodes.

The basic idea of our model is that within each slot (i.e., $[(2^i - 1) * T_0, (2^{i+1} - 1) * T_0)$) in the tilted time frame model, histograms are used to summarize the object flow pattern for each business neighbor for the period of time represented by the slot. The height of each bar in the histogram represents the volume of object flow from/to a specific node. Using this model, we can calculate the probability of an object being from/to a specific neighbor at a given time, based on the statistics.

The time representing "now" is 0, and the more distant the time is from "now", the larger the number is. Although this is a little counterintuitive, it is easy for calculation. It should be noted that the distant slot might contain some

Symbol	Description
w_e	The width of an event cycle
s_i	The i^{th} slot in the tilted time frame
b_i	The i^{th} neighbor
h_{ij}	The histogram for b_j in slot s_i
f_i	The frequency of object flow for neighbor b_i in a new event cycle
w_s	The maximum number of Exponential Event Cycles (EECs) in the slots. It is a constant.
m	The size of the reservoir sample
n	The number of business neighbors
l	The number of slots in the model (i.e., the length of the model)
T_i	The time represented by the i^{th} slot

Table 4.1: Symbols in The Overview of TISH

nodes which do not exist in the most recent one (e.g., v_4 in Figure 4.3). This is because not all the neighbors are sending/receiving objects at present, although they may have sent/received objects before.

The symbols used in the following discussion are summarized in Table 4.1.

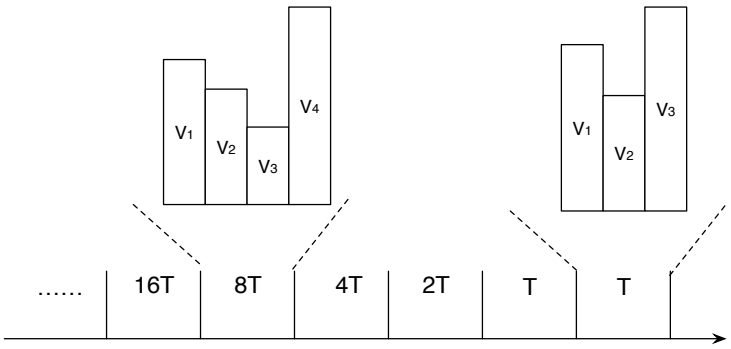


Figure 4.3: An Example of Tilted Time Frame Series of Histograms

We use source (resp., destination) LTTF to record histories for the source (resp., destination) nodes. The time series are split into *Event Cycles* (also known as the *Sliding Windows*) of fixed time interval . All the slots manage a number of (at most w_s) units, which we name as *Exponential Event Cycles (EEC)*, because

the unit in the i_{th} slot (s_i) manages the compressed data for 2^i event cycles. The most recent slot s_0 maintains the uncompressed event cycles, while the others maintain compressed ones. We denote the j^{th} EEC in the slot s_i as EEC_{ij}. It is easily inferred that $T_0 = w_s * w_e$ and the time covered by slot s_i is $T_i = 2^i * T_0 = 2^i * w_s * w_e$.

Figure 4.4 shows an example structure of the slot s_i in LTTF. Slot s_i represents the data of the time interval $[(2^i - 1) * T_0, (2^{i+1} - 1) * T_0)$. It is split into w_s EECs. In this example, w_s is 3. For each EEC, we maintain a histogram which models the distribution of data from/to a neighbor. The current size of a slot is defined as the number of used EECs in it, so $s_i.size \leq w_s$.

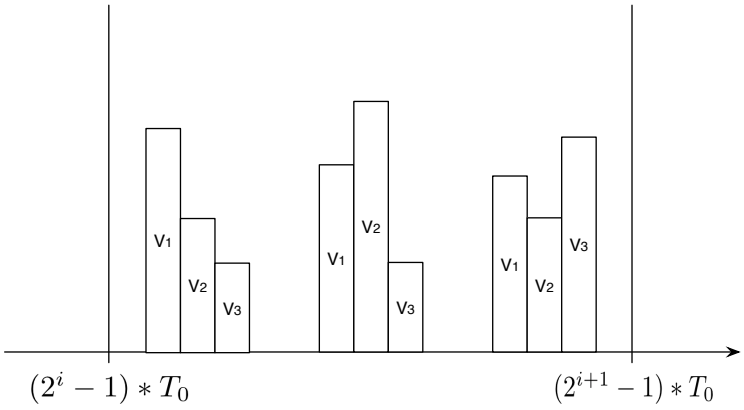


Figure 4.4: The Structure of a Slot in TISH

The reason why we split the slots further into EECs and maintain histograms for each of them, instead of maintaining one histogram for each slot, is because we want a finer control over the memory usage. By splitting the slots, we can limit the memory usage for each slot by changing the value of w_s. The larger w_s is, the more memory the model requires, and the more accurate the model is. In an extreme case, when w_s is 1, the slots are not split and the accuracy is the lowest.

The height of the bars in the histogram of each EEC is the volume of objects

flow from/to a neighbor for the time represented by that EEC. The x-axis of the histogram represents the neighbors. The *length* of the model is defined as the number of slots within it. A model of length l can store the history of $w_s * w_e * \sum_{i=0}^{l-1} 2^i$. We can calculate the length of the model which stores the history for the past t time units using:

$$l = \log_2 (\frac{t}{w_s * w_e} + 1) = \log_2 (\frac{t}{T_0} + 1) \tag{4.3}$$

Suppose we define the width of an event cycle as an hour and for each slot, there are 24 EECs with in it, i.e., w_e is an hour and w_s is 24, we only need $\lceil \log_2 365 \rceil = 9$ slots to store the history of one year. This efficiency in space enables the storage of these slots in main memory.

In the following sections, we will introduce the algorithms to sample the input streams, retrieve the source/destination data information and use such information to support the model.

4.3.2 Algorithm : RFID Stream Sampling

At node v_i, we only have the local information of the object sent to v_i, such as arrival time and departure time. To get the information about the object's moving path, it is necessary to query its neighbors. In the worst case, the underlying unstructured P2P overlay is used to locate the object. Because P2P queries are more expensive than direct combinations, we need to avoid such a case as much as possible. In addition, due to the large volumes of RFID data in the stream, we cannot afford building the model using all the data. Instead, we use a sample of the original data in each event cycle for the stream. In this section, we introduce the sampling algorithms.

Random sampling can be used as a summarization technique to capture the essential characteristics of a data set. The problem of random sampling can be informally defined as "*select a random sample of m records out of a pool of x records*". In data streams, x is unknown. This makes all algorithms that require scanning the data more than once infeasible. The reservoir sampling algorithm [Vit85] makes only one pass over the data set without knowing its size beforehand. So it is well suited for data streaming sampling. Its output is a uniform sample of the given data set. There are also some other sampling methods such as [AGP00] and [BCD03] that produce non-uniform samples for a specific type of queries. In our work, the reservoir algorithm is used because the samples are only used to maintain the histograms in the model introduced in Section 4.3.1.

The key of reservoir sampling is that for the $k_{th}(k > m)$ record in the data set, the probability of keeping it and replacing one of the record in the *reservoir* is m/k, where m is the size of the reservoir. The main feature of the reservoir algorithm is that it guarantees that the *reservoir* is always a true random sample of the data seen so far. A more in-depth discussion of this algorithm is beyond the scope of this work. Interested readers are referred to [Vit85].

Before the data stream is sampled, it should be preprocessed (see Figure 4.1). This introduces the problem of out-of-order data. In the raw data stream produced by each RFID reader, the data is sorted by the timestamp in the record. However, after the preprocessing (including cleansing, filtering etc.), the order might not be retained. Moreover, the preprocessor may join data streams from different RFID readers together. Thus the data records are interleaved (again the order is not guaranteed). Fortunately, the order of records does not influence the result of sampling, so it is unnecessary to sort the records. The reason is

that reservoir algorithm guarantees that the *reservoir* is always a true random sample, and therefore changing the position of a record in the set will not affect its probability to be included in the final sample.

4.3.3 Algorithm : Update for the Current Slot

After the data, which are collected during an event cycle, have been preprocessed and sampled, the sample is sent to the modeler for the local model update. We use the P2P tracking and tracing algorithm which we will introduce in the Section 4.4. We call the information of source and destination nodes for all the objects in the sample as *Flow Synopsis*. This synopsis is then added to the most recent slot (s_0). If the slot is full, it is moved to the slot before it (s_1). Otherwise, s_1 is summarized and merged into s_2, and then s_0 is compressed and stored in s_1. If s_2 is also full, the summarization and merging process repeats until we find a non-full slot, or we reach the end of the LTTF. In the latter case, a new slot is created and appended to the tail.

The algorithm for updating the model is described in Figure 4.5. First we add the synopsis from the new event cycle to the slot (line 1–7). If a new neighbor joins, a new entry is inserted into the hash table (line 4). If an existing neighbor did not send anything, it is set to zero (this is not shown in the figure).

If the new event cycle fills the most recent slot s_0, all slots s_i are merged if necessary (the *merge* function in line 10, this is introduced in Section 4.3.4). s_0 is then cleared to be ready for the coming event cycles (line 11).

Algorithm 1 : Update the Model: *update*

Input: Neighbor set $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$. Corresponding frequency set $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$
Output: The refined model $\mathcal{M}$

1: **for** b_i in $\mathcal{B}$
2: Gets its histogram $h_{0i} \leftarrow s_0[b_i]$
3: **if** h_{0i} is nil
4: $H_{0i} \leftarrow$ new array with size w_s, $s_0[b_i] \leftarrow h_{0i}$
5: **end if**
6: $h_{0i}[s_0.size + 1] = f_i$
7: **end for**
8: $s_0.size \leftarrow s_0.size + 1$
9: **if** s_0 is full
10: merge$(s_0, \mathcal{M})$
11: clear s_0
12: **end if**

Figure 4.5: Algorithm to Update the LTTF Model

4.3.4 Algorithm : Merging with the Next Slot

When the new event cycle fills the most recent time slot s_0, the second most recent slot s_1 will be merged to the succeeding slots recursively.

The *merge* algorithm first checks whether the next slot in the model is full. If so, it will be merged (line 4). This process is done recursively until either a non-full slot is found or all the existing slots are full. In the latter case, a new slot is created and appended to the model. Then the slot being merged will be integrated into the first non-full slot (line 5 – 14). It is compressed by merging two consecutive EECs into one (line 12). Note that in this algorithm, we assume that the width of slots w_s is even. This assumption does not affect the performance of the algorithm.

4.4 Building TISH in a P2P Fashion

The TISH model and its maintenance have been introduced in Section 4.3. In this section, we answer the unresolved question "how the model finds the source

Algorithm 2 : Merge the Model: *merge*

Input: The slot which is being merged s_i and the model $\mathcal{M}$
Output: The merged model

1: **if** s_{i+1} does not exist in $\mathcal{M}$
2: $s_{i+1} \leftarrow$ new slot, $\mathcal{M}$.append(s_{i+1})
3: **end if**
4: **if** s_{i+1} is full, merge(s_{i+1}, $\mathcal{M}$)
5: **for** each neighbor b_j in s_i
6: $h_{i+1,j} \leftarrow s_{i+1}[b_j]$
7: **if** $h_{i+1,j}$ is nil
8: $s_{i+1}[b_j] \leftarrow h_{i+1,j} \leftarrow$ new array size of w_s
9: **end if**
10: move $h_{i+1,j}[1] \sim h_{i+1,j}[w_s/2]$ to $h_{i+1,j}[w_s/2+1] \sim h_{i+1,j}[w_s]$
11: **for** $k \leftarrow 1$ to $w_s/2$
12: $h_{i+1,j}[k] \leftarrow s_i[b_j][2*k] + s_i[b_j][2*k+1]$
13: **end for**
14: **end for**

Figure 4.6: Algorithm to Merge the LTTF Model

and destination node of an object?" by introducing the P2P tracking and tracing algorithm. We also introduce a *Business Neighbor Tree* structure to ensure the stability of the whole system.

4.4.1 Tracing and Tracking Objects

With a centralized setting, like the one we introduced in Section 4.2, answering tracing queries is easy. However due to privacy and performance issues, it is impractical to use in real applications. In a fully distributed, federated environment, our model avoids using Discovery-Service-like index or flooding the whole network. The idea is to utilize the history maintained in the model to rewrite the query to the node which has the most possibility to be the source of the requested object. The tracing algorithm is defined in Figure 4.7. Tracking is almost the same except the direction is reversed to tracing, and instead of retrieving all the nodes on the object's moving path, only the latest one is retrieved.

The key idea is to query the neighbors about the object(s) in the order of the

Algorithm 3: Trace an Object: *trace*

Input: The object to trace o
The query initiating node n
Output: A list of nodes that o has been, sorted by time

1. locate any node that has had o using P2P overlay
2. $t_{start} \leftarrow$ select *Start* from Record where Id=o
3. **if** t_{start} is nil, return
4. $s \leftarrow$ the index of slot which covers t_{start}
5. $\mathcal{B} \leftarrow$ the set the neighbors in slot s and adjacent slots adjacent slots : c_1 to the recent and c_2 to the distant
6. $\mathcal{P} \leftarrow$ an array of size $\mathcal{B}$
7. **for** each neighbor b_i in $\mathcal{B}$
8. $\quad \mathcal{P}[i] \leftarrow p_1(b_i, s)$ // Equation 4.4
9. **end for**
10. $sort(\mathcal{P}, \mathcal{B})$ // sort $\mathcal{B}$ in descending order according to $\mathcal{P}$ values
11. sequentially ask all the nodes in $\mathcal{B}$ that whether it is source or destination node of o
12. upon receiving the query, b_i repeats 2–11 on itself if it had o otherwise, return a negative result.
13. b_i then sends message to n confirming the appearance of o with extra information including arrival/leaving timestamps, and application-specific data

Figure 4.7: Algorithm to Trace an Object

probability calculated according to Equation 4.4 where $p_1(b_j, s)$ is the probability of the given object coming from neighbor b_j, s is the slot covering the given time t, n is number of neighbors, c_1 and c_2 are two constants that we use to control the range of past and future data under consideration, respectively. It should be noted that using more history data (i.e., larger c_1 and c_2) does not increase the accuracy. This is because the flow patterns are unknown and may frequently vary, and as a result using more history data may add more bias.

$$p_1(b_j, s) = \frac{\sum_{i=min(s-c_1,0)}^{max(s+c_2,l)} \left(\frac{1}{2^{|i-s|}} * \frac{\sum_{k=1}^{w_s} h_{i,j}[k]}{\sum_{l=1}^{n} \sum_{k=1}^{w_s} h_{i,l}[k]} \right)}{\sum_{i=min(s-c_1,0)}^{max(s+c_2,l)} 2^{|i-s|}} \tag{4.4}$$

It is easy to get s from the requested time t, because

$$s_0.size + w_s * \sum_{i=1}^{s-1} 2^{i-1} \leq t < s_0.size + w_s * \sum_{i=1}^{s} 2^{i-1} \tag{4.5}$$

Thus,

$$s = \lfloor \log_2 \frac{t - s_0.size}{w_s} + 1 \rfloor \tag{4.6}$$

In essence, p_1 is an approximation of $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$ defined in Section 4.2. The calculation of p_1 only involves a very small portion of the data, thus it is efficient.

The tracing algorithm is shown in Figure 4.7. The neighbors are sorted (line 10) by the probabilities calculated using Equation 4.4. Then the query is redirected to the neighbor with the highest probability (line 11). If the neighbor does not return the positive result, the second possible neighbor is queried, and so on. Note the query contains the timestamp returned from previous queries, and the neighbor only returns positive result if the timestamp is earlier than the one in the query. Otherwise, an infinite loop may happen if the object visited a node more than once.

This algorithm is an online process. It does not return the result immediately or in real time, but generates the result (which is a list) gradually over time. The sacrifice of timeliness brings space efficiency and privacy. In addition, with our model, P2P queries can be avoided so the time complexity is almost optimized.

4.4.2 Building the Flow Synopsis

After the data within an event cycle is sampled, we need to ask the neighbors to get the source nodes. Instead of flooding the network, the history of object flow (i.e., the histogram) is used to find the most possible neighbors who may be the source nodes. Assuming that the object flow pattern changes smoothly, we choose the recent slots in the model to compute the probability of a neighbor being the source node for the objects. Instead of only using the most recent slot,

we introduce a zipf-weighted method to compute the probability with several recent slots. This mechanism is introduced to smooth the data flow in case that there are some peak moments for a neighbor. Equation 4.7 shows how the probability is computed. It is easy to see that $p_2(b_j)$ is a variant of $p_1(b_j, s)$ that we introduced in Section 4.4.1. Essentially, flow synopsis building is a simplified version of tracing. The differences between the two are:

1. Flow synopsis building happens on a node which must have the objects, whereas tracing does not have this advantage so it has to invoke the underlying P2P layer to first locate a node having had the object being traced.

2. Flow synopsis building happens when objects have arrived at a new node. So there is no future information. When calculating the probability, there are no future slots. In contrast, when answering tracing queries, future data has to be included.

3. There is no time or slot parameter for p_2, because p_2 is used to calculate the probability of a neighbor being the source node for objects in the *most recent* slot, i.e., the implicit time parameter is "now".

$$p_2(b_j) = \frac{1}{\sum_{i=0}^{c} 2^i} * \sum_{i=0}^{c} (\frac{1}{2^i} * \frac{\sum_{k=1}^{w_s} h_{i,j}[k]}{\sum_{l=1}^{n} \sum_{k=1}^{w_s} h_{i,l}[k]}) \tag{4.7}$$

Where c is a configurable constant (similar to c_2), representing the number of slots under consideration, and n is the number of business neighbors. The factor $\sum_{i=0}^{c} 2^i$ is for normalization. $\sum_{l=1}^{n} \sum_{k=1}^{w_s} h_{i,l}[k]$ calculates the total number of objects for a slot, while $\sum_{k=1}^{w_s} h_{i,j}[k]$ is the number of objects from b_j. The factor $\frac{1}{2^i}$ assigns weights to different slots. The most recent slot has the highest weight of 1, and the weight decrease exponentially. In implementation, the sum

of frequencies for all nodes in a slot ($\sum_{l=1}^{n}\sum_{k=1}^{w_s} h_{i,l}[k]$) can be calculated and cached.

With Equation 4.7, the neighbors can be sorted by probability p_2. We can first try to contact the neighbor with the highest probability. If there are objects without source node after this query, the neighbor ranked the second is queried. The process repeats until we find the source nodes for all objects in the sample, or we have tried all the neighbors. For the latter case, if there are still objects without source node, we will have to rely on the P2P overlay to locate the object.

When a node receives the request to verify whether it is the source node for a list of objects, it not only responds, but also updates the model for the destination nodes, if the requested objects, or part of them, are from itself. The update algorithm is exactly the same as the one to update the source LTTF, which is introduced in Section 4.2.

Figure 4.8 shows the details of the algorithm to gather the source node information for the TISH model. First the probabilities $\mathcal{P}$ for all the neighbors are calculated and the neighbor list is sorted according to the probabilities (line 1 –4). Then we query the neighbors for the list of objects with unknown sources, in descending order of sorted probabilities (line 6–11). After querying the neighbors, if there are still objects with unknown sources, we have to use underlying P2P overlay to find the sources of the objects (line 12–18). Finally, the neighbors and the corresponding frequencies are used to update the TISH model (line 19).

Note that the frequency is scaled (at line 8, 15) to the size of the original data set. The reason is that we need not only the object distribution between different neighbors, but also the volume changes of the object flow from the same neighbor.

Algorithm : Building Flow Synopsis

Input: A set of sample objects $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$
A set of neighbors $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$
Number of objects in the unsampled event cycle: x
Output: The updated sender LTTF model

1: $\mathcal{P} \leftarrow$ an array of size n
2: **for** each neighbor b_i in $\mathcal{B}$
3: $\mathcal{P}[i] \leftarrow p_2(b_i)$ // Equation 4.7
4: $sort(\mathcal{P}, \mathcal{B})$ // sort $\mathcal{B}$ in descending order according to $\mathcal{P}$
5: $\mathcal{F} \leftarrow$ a map from b_i to its frequency
6: **for** each neighbor b_i in $\mathcal{B}$
7: result set $R \leftarrow query(b_i, \mathcal{O})$
8: $\mathcal{F}(b_i) \leftarrow R.size/m * x$
9: $\mathcal{O} \leftarrow \mathcal{O} - R$
10: **if** $\mathcal{O} = \Phi$, break
11: **end for**
12: **if** $\mathcal{O} \neq \Phi$
13: **for** each object o_i in $\mathcal{O}$
14: $b \leftarrow P2POverlay.locate(o_i)$
15: **if** b exists in $\mathcal{M}$, $\mathcal{F}(b) \leftarrow F(b) + x/m$
16: **else** $\mathcal{F}(b) = x/m$; $\mathcal{B}.append(b)$
17: **end for**
18: **end if**
19: $update(\mathcal{B}, \mathcal{F})$

Figure 4.8: Algorithm to Build Flow Synopsis

In real applications, it is not often that new neighbors join. So in most cases, underlying P2P queries do not happen. In addition, since the change of patterns is infrequent in most applications, normally we only need to query a few neighbors till we have all the objects' source nodes. Nevertheless, this algorithm is very sensitive to the changes of patterns or the network. For example, if a new neighbor v_j starts sending objects to v_i, in the first round v_i has to use P2P queries to find out that v_j is a source node. However, because we assign higher weights to the recent data, v_j will have high ranking in the candidate list for the coming event cycles.

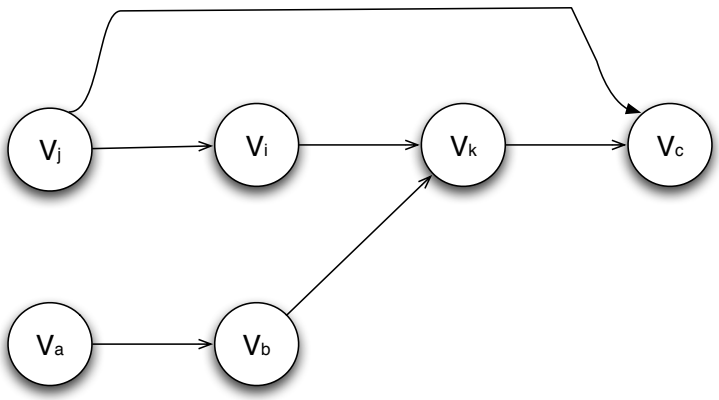


Figure 4.9: An Example of Sideway Problem

4.4.3 The Business Neighbor Tree

There are two critical issues in real-life applications. Firstly, a node may go offline with or without notification to other nodes in the network (but objects are still moving from/to that node). In this case, the information of connections relevant to that node becomes unavailable. Secondly, a node v_i and one of its source nodes v_j may both be the direct source node of v_k. We call this situation the *Sideway Problem*. Due to this, the algorithms in Figure 4.8 and Figure 4.7 may fail to get the actual source node. As shown in Figure 4.9, suppose object o moved from v_j to v_i, then from v_i to v_k. Both v_i and v_j are in the v_k's business neighbor set, and for the past few event cycles, v_j has more objects moving to v_k. Since v_j is the first to be queried, it will be treated as the source node of object o from which it moves to v_k. Clearly, this is not correct because the source node should be v_i.

A source tree at a node v_k is a subgraph of the whole network topology, with v_k as the root, representing the topology of nodes that have either direct or indirect connections to v_k. Figure 4.10 depicts the source tree maintained at v_k in the sub-network shown in Figure 4.10. The root of the tree is v_k. Its child nodes are the *direct* source nodes of v_k (i.e., v_i, v_j and v_b in Figure 4.9).

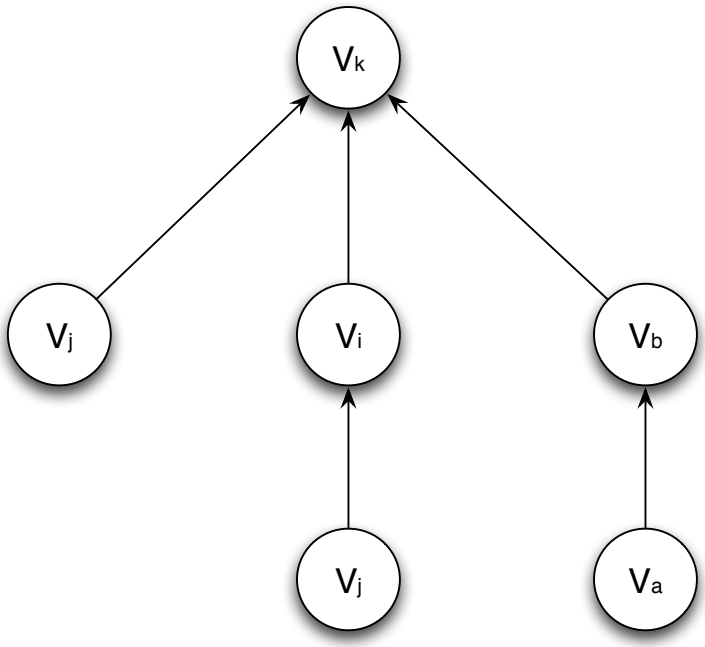


Figure 4.10: An Example of Business Neighbor Tree

The maintenance of the source tree is done simultaneously with the flow synopsis building. In the algorithm to gather the source node information (Figure 4.8), neighbors also include its source tree in the result set R. This source tree is then merged with the local source tree. In this way, the source tree is built recursively.

This structure adds more connectivity to the network, which increases the stability of the system in the case that some nodes may leave. However, it does not require replication of any form. It also helps to solve the *Sideway Problem*.

Solving Node Leaving Problem. With the source node tree, when a neighbor v_i leaves the network, we can query its direct source nodes $\mathbf{S}(v_i)$ to check whether an object comes from v_i, because if an object comes from a node's direct source node v_i, it must also come from one of v_i's direct source nodes. In the source node tree, the LTTFs for all the nodes in $\mathbf{S}(v_i)$ are maintained, together with

the LTTF for v_i itself. If an object comes from a node v_a in $\mathbf{S}(v_i)$, it is used in the maintenance of the LTTFs for both v_i and v_a. In this way, even when v_i is offline, its LTTF is kept accurate.

We do not remove v_i from the tree when it leaves. Instead, we mark it as "disconnected". After it is back online, we remove the "disconnected" mark. This is useful to deal with temporary departure of nodes (e.g., short server breakdown). The LTTFs for nodes in $\mathbf{S}(v_i)$ are deleted because we no longer need them to build the flow synopsis. In this way, even when some nodes in the network leave, we can still respond to tracing queries and continue to build flow synopsis.

If after a certain configurable time, v_i is still offline, it will be removed from the tree and its LTTF will be deleted or archived.

Solving Sideway Problem. To solve the *Sideway Problem*, for the algorithms to answer tracing queries and building the flow synopsis, if a node has both direct and indirect connection to the root node, it will be queried regardless of the order of probabilities. For example, in Figure 4.10, not only v_j is the direct source node of the root node v_k, but there is also an indirect connection $(v_j \rightarrow v_i \rightarrow v_k)$ between v_j and v_k. v_j is queried regardless of its order in the sorted neighbor list.

After receiving the result set, we compare the arrival timestamps of the objects which are included in the result sets from more than one neighbors and select the node with the latest timestamp as the source node. For example, if the arrival timestamp for an object is earlier in v_j than that in v_i, it is clear that the object was sent through the path $v_j \rightarrow v_i \rightarrow v_k$.

4.4.4 Determining w_s and w_e

Generally, our model is a synopsis for the RFID stream. A generic description of such a data structure is that it supports two operations, *update* and

computeAnswer [BBD02]. In our model, the *update* operation takes more time than *computeAnswer*, because it involves network queries). More importantly, it may be so slow that the histograms for the past event cycle have not yet been constructed when it starts constructing the histograms for the current event cycle. In this case, the most recent histograms in slot s_0 is actually an old one. If this happens, the most valuable data is not used because it is not ready yet. Figure 4.11 illustrates such situation. At the end of the second event cycle (EC2), the construction of histograms for EC1 has not been done (which will be done at $w_e + t_u$, where t_u is the time used for update).

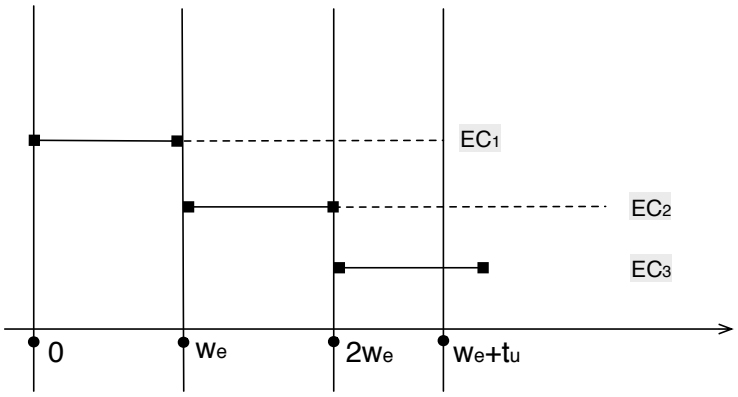


Figure 4.11: An Example of Overlapped Sliding Windows

To avoid this problem, w_e should be large enough, i.e., w_e should be larger than the maximum possible value of t_u. But it should not be too large. If it is too large, the pattern of object flow may change during an event cycle. In this case, the change is not captured. t_u is determined by the size of the sample, the number of the business neighbors and the size of the whole network as discussed in Section 4.2. It is a dynamic value. A good way to determine w_e would be to make it a function of the three parameters. However maintaining the size of the whole network is not an easy task, and is costly. In this work, we propose an

adaptive approach.

When a node joins the network, it sets w_e to a default value. During the maintenance process of the model, if ${t_u}^2$ becomes larger than w_e or smaller than $\frac{w_e}{2d}$, $w_e \leftarrow d * t_u (d > 1)$ (d is a constant between 1 and 2). In real applications, after the network becomes stable, t_u will likely become stable too, so w_e is almost a constant. Our algorithms, which are based on the assumption that w_e is a constant, are not affected. To deal with the unstable phase of model construction, each event cycle in the model is parameterized with a timestamp t_{end} for the end of the cycle.

The maximum number of EECs in a slot (i.e., w_s) has the influence on the memory usage of the LTTF model. Generally, the memory usage of the model is $O(l * w_s * n)$ (l is the length of model and n is the number of neighbors). w_s can be large, as long as the memory is sufficient. Larger w_s ensures the model with a finer granularity.

4.5 Performance Analysis

4.5.1 Model Maintenance Cost

The time complexity for maintaining the TISH model consists of three parts: i) sampling the input, ii) querying the source nodes, and iii) updating the model. Suppose t_u is the time used to update the model for an event cycle, we have:

$$t_u = t_{sample} + t_{query} + t_{update} \tag{4.8}$$

Using the reservoir algorithm, we only need to scan the input once, and this

[2] t_u is an observable variable.

$\mathcal{B}$

v_1	v_2	v_3	v_4	v_5
50	30	20	0	0

$\mathcal{B}'_1(\delta = 0)$

v_1	v_3	v_2	v_4	v_5
50	30	20	0	0

$\mathcal{B}'_2(\delta = 2)$

v_1	v_2	v_4	v_5	v_3
50	30	10	10	0

$\mathcal{B}'_3(\delta = 1)$

v_1	v_2	v_4	v_3	v_5
50	30	10	10	0

Figure 4.12: Example of Performance in Modeling Accuracy

can be done on-the-fly. So the cost for this part is $O(x)$, where x is the total number of objects in the current event cycle.

For the worst case, the algorithms in Figure 4.7 and Figure 4.8 will go through the whole model, with a complexity of $O(l)$, where l is the number of slots in the model. It should be noted that we can avoid the movement of histograms shown in Figure 4.7 (line 10). l is often small as we discussed in Section 4.2.

The most costly part is querying the source nodes, as the network latency is typically much longer than the local processing time. Obviously, when querying the source nodes, the queries can be sent out simultaneously. However, the network traffic cost will be $O(n)$ (n is the number of neighbors). This is not economical because building the flow synopsis is not necessary to be done in real time. Indeed, the TISH model is only updated when an event cycle ends, and the maintenance time t_u is controlled to be less than an event cycle.

With the algorithm shown in Figure 4.8, we can save the network traffic cost by querying the nodes which have most possibility to be the source node first. Moreover, we can also avoid the underlying P2P queries when there are no new business neighbors joining. Even if there is one, after the first event cycle, the new neighbor will be in the LTTF model with high probability according to Equation 4.4. As a result, the network cost is reduced.

The accuracy of the model is very important for reducing network traffic cost to maintain the model. We represent the real distribution of objects' source nodes during a specific time period as $\mathcal{B}$, which is the neighbor list sorted in descending order by the real distributions (e.g., $\mathcal{B}$ in Figure 4.12). For example, in Figure 4.12, there are five possible source nodes (v_1 to v_5) with the possibilities of 50% for v_1, 30% for v_2, and 20% for v_3. The distributions modeled by the TISH model in the same period of time is denoted as $\mathcal{B}'$. Figure 4.12 depicted different scenarios for the accuracy of the TISH model.

Suppose that, the m objects (the value of m does not matter in this discussion), which we want to query their source nodes, come from y different nodes. In this case, $y \leq m$. In Figure 4.12, y is 3, because in the real distribution $\mathcal{B}$, the m objects can only come from v_1, v_2 and v_3.

In the best case scenario, y queries are needed. Because we have to query the first y nodes in order to find source nodes for all objects. $\mathcal{B}'_1$ is one of the best scenarios. Although the order of v_2 and v_3 are not the same as $\mathcal{B}$, 3 queries are still enough to find the source nodes. In general, the best scenario happens as long as the first y nodes in the model $\mathcal{B}'$ are the same as the first y nodes in the real distribution $\mathcal{B}$, regardless of the order.

In the worst case scenario, all the nodes in the neighbor list are queried. $\mathcal{B}'_2$ is one of these cases. The neighbor v_3 which indeed is a source node, is ranked the last in $\mathcal{B}'_2$, so we have to query all the 5 neighbors to find the source nodes for all objects. In general, the worst case happens when there exists one node in the first y nodes of $\mathcal{B}$ being ranked the last in $\mathcal{B}'$.

Suppose δ is the number of *extra* queries made in addition to the optimized value y, we can conclude that:

$$\delta = max_{i=1}^{y}(\mathcal{B}'.rank(\mathcal{B}[i])) - y \tag{4.9}$$

For example, in Figure 4.12, δ is 0 for $\mathcal{B}'_1$ because all first 3 nodes in $\mathcal{B}$ are still ranked first 3 in $\mathcal{B}'_1$, thus $max_{i=1}^{3}(\mathcal{B}'_1.rank(\mathcal{B}[i]))$ is 3. For $\mathcal{B}'_2$, v_3 in first 3 of $\mathcal{B}$ is ranked 5 in $\mathcal{B}'_2$, thus $max_{i=1}^{3}(\mathcal{B}'_2.rank(\mathcal{B}[i]))$ is 5, so δ is 2 (i.e., 5 - 3). Similarly for $\mathcal{B}'_3$, v_3 is ranked the 4^{th} in $\mathcal{B}'_3$, so δ is 1. The goal of our model is to keep the average value of δ as small as possible.

4.5.2 Performance of Building Flow Synopsis

The cost of tracking and tracing consists of two parts. If the initiating node of the query is not on the movement path of the object, the query should first be redirected to a certain node on the path. The cost of this step is the cost of the underlying P2P overlay lookup cost. However, in real applications, it is rare that the query is initiated by a third party. So in most cases, this part of the cost does not exist.

The second part is the query cost. The total cost of a trace query (t_{trace}) is a function of the length of the object's moving path. In general, $t_{trace} = \sum_{i=1}^{l} t_i$, where t_i is the time used to establish the i^{th} segment in the path and l is the length of the path. We assume that t_i is proportional to the number of queries (q) made, so $t_i = q * \mathbf{T}$ where $\mathbf{T}$ is a constant representing normal remote query processing time. Clearly, the cost is determined by q. It is easy to infer that if the object comes from the j^{th} neighbor in $\mathcal{B}$, then j queries are needed to find it. Thus, the optimized average value of q is (in this section, we reuse the definition of notations in Section 4.5.1.):

$\mathcal{B}[j]$	probability	Number of Queries			
		$\mathcal{B}$	$\mathcal{B}'_1$	$\mathcal{B}'_2$	$\mathcal{B}'_3$
v_1	0.5	1	1	1	1
v_2	0.3	2	3	2	2
v_3	0.2	3	2	5	4
q or q'		1.7	1.8	2.1	1.9

Figure 4.13: Examples of Tracing Efficiency

$$q = \sum_{j=1}^{y} (j * \mathcal{B}[j].probability) \tag{4.10}$$

i.e., the order of the querying follows the *actual* order of the neighbors sorted by the probability that the object comes from them.

The number of queries q' by using the TISH model is:

$$q' = \sum_{j=1}^{y} (\mathcal{B}'.rank(\mathcal{B}[j]) * \mathcal{B}[j].probability) \tag{4.11}$$

Figure 4.13 shows an example of the optimized cost and the cost with the TISH model by using the examples in Figure 4.12. If the actual order ($\mathcal{B}$) is followed when querying the source node, we only need, on average, $0.5*1+0.3*2+0.2*3 = 1.7$ queries. But in a different order, for example, in $\mathcal{B}'_1$, the order of v_2 and v_3 is reverted. In this case, we have to use $0.5 * B'_1.rank(v_1) + 0.3 * B'_1.rank(v_2) + 0.2 * B'_1.rank(v_3) = 0.5 * 1 + 0.3 * 3 + 0.2 * 2 = 1.8$ queries.

The goal of tracing query processing is to make q' as close to q as possible.

4.6 Related Works

In [CKR04b] and [DOL07], the fundamental problems in RFID data management and query processing are discussed. One of the important topics lies in how to

develop an efficient model to infer the implicit business knowledge from large volumes and distributed RFID data streams. In this section, we review the major techniques that are most closely related to our proposed approach.

4.6.1 Data Structures and Data Transformation

A raw RFID record is a triple tuple $(ID, time, location)$ [CKR04b], which presents little value until it is transformed into a form suitable for application-level interactions. In addition, such data has implicit meanings and associated relationships with other RFID records where applications have to make appropriate inferences [SLZ08a].

In one of the earliest efforts on RFID data modeling, Wang et al. propose an RFID data model that abstracts static and dynamic entities including object, reader, location and transaction [WL05b, WLL10] . It models the interaction between them as either *state* or *event* based relationships. The data model also provides a rule-based data filter engine. In [LWL06], RFID applications are classified into a set of typical scenarios and a generalized data modeling framework with constructs for each typical scenario is proposed. In [HSC05], Hu et al. focus on the path encoding by using a Bitmap data type and they demonstrated that significant storage savings can be achieved. In [LC08], a novel data structure and algorithms to efficiently encode, decode and query an object's moving path in an RFID database are proposed. The approach has been further improved very recently in [LC11] to cope with the situation where the moving path is long. Finally, Lin et al. propose in [LEB07] a "Multi-Table" model in which *path* and *containment* relationships can be defined.

4.6.2 Knowledge Discovery

For an RFID data warehousing approach proposed by Gonzalez et al. in [GHL06b], the authors observe that individual objects tend to move and stay together (i.e., bulky object movements in supply chain). They propose a novel model to compress RFID data without information loss. The model consists of a hierarchy of highly compact summaries of data aggregated at different abstraction levels where analysis takes place. These summaries are represented as RFID-cuboids. Each RFID-cuboid records object movements and stores product information for each RFID object, information on objects that stay together at a location, and path information necessary to link multiple stay records. This work is further improved in [GHC10] by discovering the *Gateway* nodes that have either high fan-in or high fan-out edges. The RFID cuboids are created based on this discovery to save more space. This method is restricted to process static data in centralized environments. It is not suitable for processing RFID data streams.

In [LP08], Lee and Park propose a dynamic tracing model for supply chain management, which considers not only the movements, but also the combination and splitting of RFID-attached objects. In [NUY10], an algorithm to extract frequent sequential patterns from RFID data streams is proposed. The authors also introduce a decision tree model to determine the prime movements of tagged objects by using the extracted patterns. TMS-RFID [LLSar] is a system to manage temporal RFID data, which extracts complex temporal event patterns over RFID streams. In [GYG09], the authors propose a probability evaluation model and algorithms for moving range query processing. These models focus on different aspects of modeling RFID data, but they all have the same limitations as RFID-cuboid.

SPIRE [NCCar] is a novel RFID data interpretation and compression system.

It extracts location and containment relationships over RFID streams by applying a probabilistic algorithm over a time-varying graph model. This model can be deployed in either centralized or distributed environments. However, it still requires full access to all the data.

The BRIDGE project [3] is an implementation of the EPCglobal framework, whose features and limitations will be discussed in the next section. It includes a probabilistic model to predict future location of RFID-attached objects using Markov Chain. However, this model is designed specifically for supply chain management systems. It requires synchronization with static supply chain model, which makes it inflexible.

4.6.3 Distributed Modeling and Query Processing

Due to the large volume of RFID data streams, centralized solutions are often infeasible in large-scale RFID applications, especially those involving several organizations/companies. Researchers have invested a lot of efforts to develop new distributed systems or adapt the centralized ones for distributed environments.

EPCglobal[4] is an organization focused on developing standards to support RFID in information rich trading networks. It has developed a *Discovery Service* standard which is used to trace individual items. To enable the traceability, partners have to register all the objects to the service. This architecture is not fully distributed and scalable.

The authors of [CSD11] extend their work on SPIRE [NCCar] to adapt to large-scale RFID networks. The location and containment relationships are inferred in a distributed way. [ACK06b] proposes a pure distributed RFID data

[3] http://bridge-project.eu
[4] http://www.epcglobalinc.org

model. Two attributes *sentTo* and *receivedFrom* are associated with each object. The distributed path is formed by records in correlated nodes. However, this work does not solve the problem on how to acquire these attributes. Sheng et al. solve this problem by using a DHT-based architecture [SWR10]. This work requires every item to be indexed in the network which makes the approach costly. This same effort is further extended in [WSR11a] by introducing a model which indexes the objects in a structured P2P network and algorithms to maintain the model. However, this model supports item-level and aggregation traceability queries along with the cost of indexing spaces.

Query processing in a P2P environment is essentially searching for the proper resource to answer the query. The general P2P architectures, such as [LXN07], [BJ01] and [Sto01], can be applied. However, RFID records have implicit knowledges about the distribution of objects, which can direct the search in a more efficient fashion than the general methods. In [KCW09], Jinoh et al. proposes the notation of accessibility to capture both availability and performance as a measurement in node selection. However, this work is still too generic to consider the data itself as a reference. Most existing *Content-Aware* P2P systems, such as [CJLar] and [THI10], focus on efficient replication of the data to increase its availability based on the content of data. They are not feasible in processing RFID data streams because the partners require sovereignty of the data. In addition, compared to texts or multimedia resources, replicating RFID data is often unnecessary since only a very small portion of them is going to be queried.

Finally, in a very recent work in [AH11], the authors propose a framework for RFID-based inter-organizational cooperation. This work includes a cooperative, complex event processing method, which is based on event notification services.

4.7 Summary

In this chapter, we have introduced a distributed model and algorithms for mining sovereign RFID data streams. The main features of our approach include:

- By combining the techniques of *Titled Time Frame* and *Histogram*, we can describe the changes of moving patters in both dimensions of time and space.
- The model is purely distributed. As a result, it is scalable. We developed distributed algorithms to establish and maintain the model.
- Our proposed model and algorithms are efficient in terms of bandwidth cost and querying time. We demonstrated the usefulness of this model in processing tracking and tracing queries.

We have implemented the proposed techniques in our prototype. The details of the implementation and performance study can be found in Chapter 6.

Chapter 5

PeerTrack Cloud: An Affordable, Flexible and Scalable Architecture for Traceable RFID Networks

In Chapter 3, we introduced a P2P architecture and data model for item-level traceability in distributed RFID networks. In Chapter 4, we further proposed a distributed mining model for discovering object moving patterns from distributed RFID streams. These works solve the requirements of *Scalability*, *Privacy* and *Heterogeneity* which we examined in Chapter 2. However, due to the characteristics of P2P architecture, the *Timeliness* requirement has not been addressed. These P2P architectures are not suitable for applications under real-time constraints.

On the other hand, each node in the network is required to engage an upfront commitment of capacity for the peak of object flows[1]. As we have discussed in Chapter 4, RFID data often exhibits certain patterns in both time and space

[1]This is not a specific problem for our architecture, but also for most data intensive applications.

dimensions, which means, the peak volume appears very rare. In most of the time, the upfront commitment of hardware and bandwidth is a waste of resources.

Fortunately, the evolution of *Cloud Computing* [AFG10] has brought us the chance to solve these problems. Cloud computing eliminates the requirement for users to plan ahead for the provisioning because it provides pay-as-you-go scheme for the hardware and bandwidth. However, this is not the only benefit of cloud computing. Cloud computing can also lower the operating costs, reduce the business risks and provide easy access to developers and users [ZCB10]. These are all important factors for the success of large-scale RFID applications. Another important benefit of cloud computing is the reliability and availability of data. Cloud providers replicate the data in several data centers which are distributed all over the world [AFG10] so that the single failure of a server will not cause the interruption of the whole service. And the requests can be routed to the geographically nearest data centers so that the processing time can be optimized.

In this chapter, we first introduce a Cloud-based architecture for traceable RFID networks which takes the advantages of the aforementioned benefits of cloud computing. This architecture is built on the top of the techniques that we proposed in Chapter 3 and Chapter 4. It enables item-level traceability and object moving patterns mining. We call this architecture "PeerTrack Cloud". More importantly, we introduce the algorithm for RFID data replication so that the data for all (or most of) the nodes along a path are replicated in the same data centers which are "closer"[2] to the nodes along the given path. In this way, the traceability queries can be answered by visiting only one data center.

Our contributions are summarized as follows:

- We introduce a Cloud-based architecture for traceable RFID networks and

[2]The definition of "closer" will be introduced in Section 5.3.

expose the Traceability as a Service(TaaS) interfaces. This architecture does not require the partners in the network to plan for the provisioning beforehand. Instead, they pay the storage and bandwidth fees for the amount that they have used. This elasticity makes it affordable for more partners in supply chain networks to join the traceable network because the smaller business owners no longer need to worry about the costs for hardware and maintenance. The architecture is built on the top of the distributed traceability network and mining models that we introduced in this dissertation. However, it is not a simple data migration, but a different paradigm in system architecting.

- We develop algorithms to replicate the RFID data. Cloud providers often have some kind of replication strategy. However, these replication strategies are too generic to adopt to traceable RFID networks directly. Objects move in certain patterns in RFID networks, which generate data distributions in corresponding patterns. Based on the data model and algorithm to discover the patterns that we introduced in Chapter 4, we further introduce the moving pattern based algorithms for data replication.

- The underlying techniques make the system elastic. As a result, new nodes can join and existing nodes can leave at any time. The elasticity is very important in realizing "Internet of Things" where the network is highly dynamic.

The rest of this chapter is organized as follows. In Section 5.1, we discuss the problems in existing systems and the motivation to move large-scale RFID applications to the Cloud. In Section 5.2, we introduce the Cloud-based architecture for RFID traceable networks. We then introduce the algorithms for data

replication in Section 5.3. In Section 5.4, we analyze the performance of the Cloud-based architecture and replication algorithms and compare the storage and bandwidth costs of our architecture with other architectures. The related works are discussed in Section 5.5. Finally, Section 5.6 provides some concluding remarks.

5.1 Motivations and Challenges

The paradigm behind the existing RFID applications (including ours) is known as "Data on Network" [DMS07]. Its basic idea is that the tags attached to the objects carry only reference numbers, meanwhile the data pertaining to the objects are stored in a network of databases and they can be retrieved with the reference numbers. In this way, the data storage and the objects are physically isolated. This makes it possible to build the system in a distributed environment because it is possible to access the data without physical access to the objects.

In some applications, downstream business operations may require fast access to data records from upstream part of the object moving path. For example, a distribution center in a supply chain needs to get the destination information from the source of the delivery. When the distributing process is fully automatically operated by integrating RFID and robotic techniques, it is necessary that between the time when the object is scanned by the RFID reader and the time when it reaches the redistribution point, the destination information must be retrieved. Another example is that the downstream business partners (e.g., wholesalers and retailers) should be warned immediately when they receive the wrong products.

Existing architectures for large-scale RFID applications all feature distributed architectures (See Chapter 2). On the one hand, the distributed architectures do provide high scalability. On the other hand, they are not feasible under real-time constraints, because they often require several remote data access steps to get this information. For example, with EPCglobal architecture, we must invoke the Discovery Service (DS) first to get the list of EPC Information Service (EPCIS) instances. Then we query the EPCIS instances to get the required information. For P2P networks, the number of nodes visited in the processing of a query is $O(log_2|\mathcal{V}|)$ where $|\mathcal{V}|$ is the size of the network (see definitions of notations in

Chapter 2). These long latencies can cause problems whenever a process requires fast responses. To solve this problem, we need to find a new method which is distributed (i.e., scalable) and responsive.

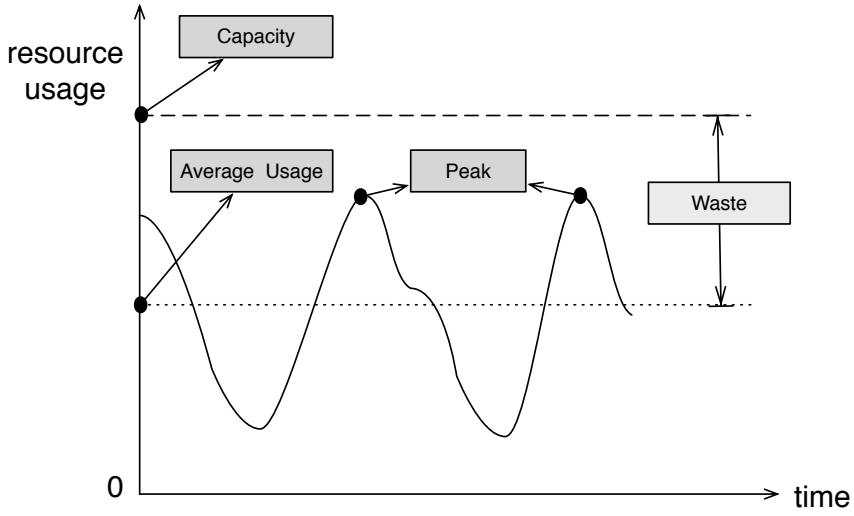


Figure 5.1: The Waste of Resources in Existing Deployment Scheme

The common deployment scheme in existing RFID applications mandates the capacity of hardware to match the peak volume of the data. This causes a waste of bandwidth and power resources. Figure 5.1 illustrates this problem. As we have discussed in Chapter 4, the movement of objects often exhibits certain patterns. More specifically, they often move in groups (e.g., in supply chain network, objects are transported together in containers). As a result, the RFID stream exhibits corresponding patterns. Existing applications require the system to run in the capacity higher than the estimated peak volume of the stream. However, the peak volumes rarely happen. Consequently, the resources are wasted by $(capacity - average\ usage) * time$. Another problem is, the peak volume can only be obtained by estimation, and it changes. For example, it is possible that the CEO of a large supermarket corporation decides to push a major sale, which leads to an *unscheduled peak*. To overcome this problem, the IT department has to rent or buy

more servers.

Cloud computing is a different deployment paradigm for large-scale applications [AFG10]. There are many definitions for cloud computing [VRC08]. In our opinion, among them, the National Institute of Standards and Technology (NIST) definition[3] covers all the essential aspects:

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

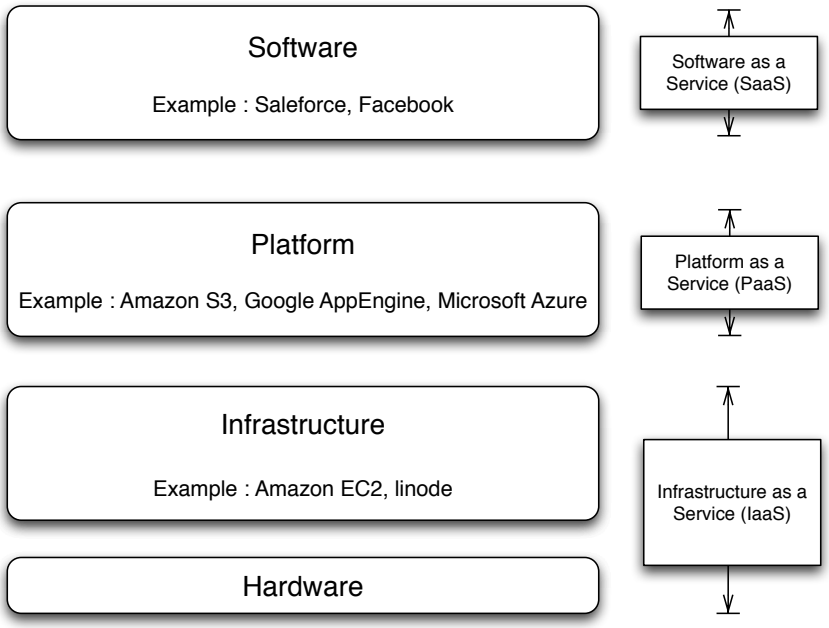


Figure 5.2: Cloud Computing Architecture

Most of the technologies used in cloud computing is not new. On the contrary, they have existed for a long time. However, cloud computing is a new operation model which brings together these technologies and provides various kinds of

[3] http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf

services (storage, computing etc.) to users in an economic way. The architecture of cloud computing environment is a layered structure, which is depicted in Figure 5.2 [ZCB10].

In cloud computing architecture, the upper layers depend on the services provided by the layer below, like in the OSI network layers. Services in each layer can be provided to end users via a set of predefined service interfaces. For example, Amazon[4] provides EC2 (Elastic Compute Cloud) virtual machines as a service in the infrastructure layer, meanwhile, it also provides Amazon S3 (Simple Storage Service) as a storage service in the platform layer. Users of these services do not need to engage an upfront commitment for their usage. Contrarily, they only need to pay for the amount that have been consumed.

Cloud computing brings a lot of benefits [ZCB10, AFG10] including but not limited to:

- **On-Demand Service** The usage of storage and network resources is fully determined by the demand of consumers. There is no upfront commitment and users can quit at any time. This advantage solves the problem in Figure 5.1.
- **Geo-distribution Access** The cloud providers build several data centers in different locations around the globe. Users will be directed to the nearest data center which can serve their requests. As a result, user experience is globally same for people in different regions/countries.
- **Shared Resource Pooling** The cloud providers offers a pool of resources and dynamically assign them to users according to their needs. The providers

[4] `http://www.amazonaws,com`

take care of the maintenance of the resources and have the chance to globally maximize the resource utilization.

There are also many other benefits of cloud computing which interested readers can find in [ZCB10, AFG10]. In our opinion, the above three are the most important features of cloud computing for traceable RFID networks. On the one hand, because the service can be provisioned on demand, we need not to estimate the peak anymore. And since the resources are shared and pooled, when the peak volume happens, the user does not need to do anything and leave all the reassignment of resources to the cloud providers. On the other hand, thanks to the geo-distributed data centers, we can redirect the requests to the nearest or most responsive data center to expedite the traceability query processing.

To successfully build a cloud-based traceable RFID network, there are two major challenges that we have to consider.

- **System Architecture** The architecture of the system must be carefully redesigned in order to make the most use of the benefits from adopting proper services from different layers of cloud computing architecture. And the system should provide the services to end users in a flexible yet powerful way.

- **Data Replication Strategy** Although cloud providers replicate data in several data centers for geo-distribution and better availability, the replication strategy is not designed specifically for RFID data. RFID data at different nodes have closer relationship than regular data. For example, the downstream node often requires to read the data from upstream node and two adjacent nodes are often close to each other geographically. It will expedite the traceability query processing if the data for all related nodes

in a path are already stored in the same data center.

In the next two sections, we will introduce our cloud-based traceable RFID network architecture and a novel data replication algorithm.

5.2 The Architecture of PeerTrack Cloud

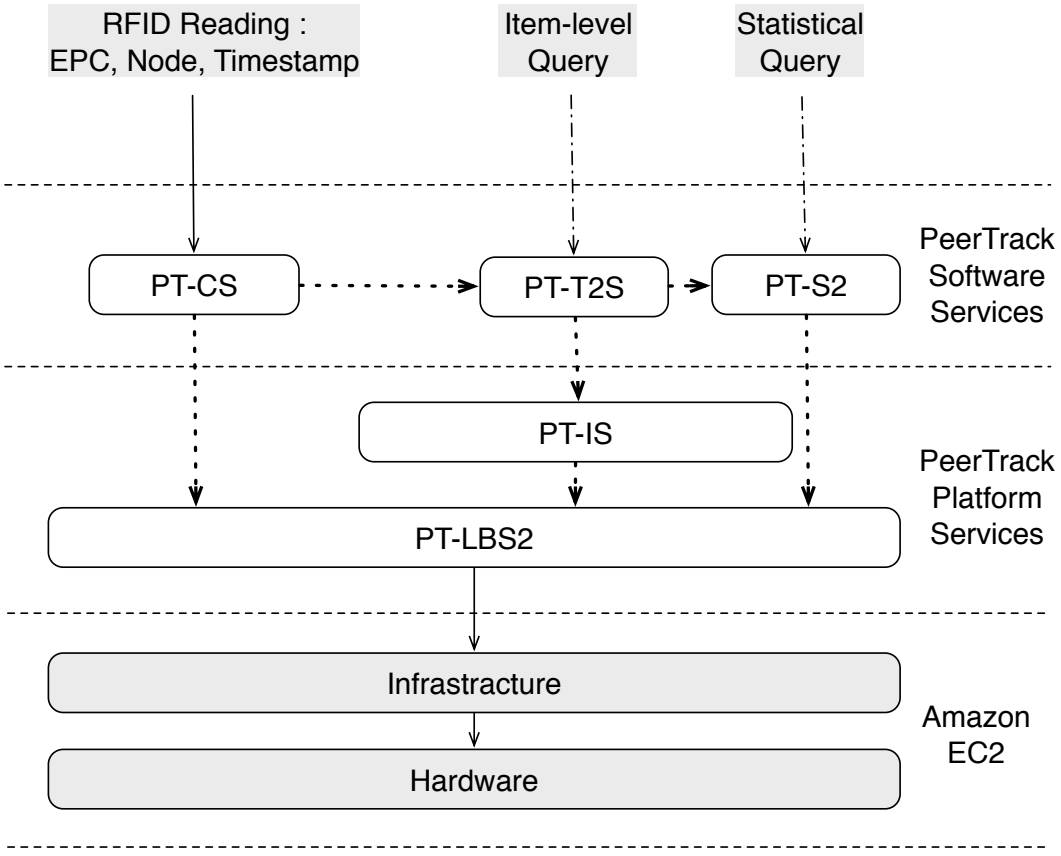


Figure 5.3: PeerTrack Cloud Architecture

5.2.1 Modules of the Architecture

The architecture of the PeerTrack Cloud consists of five major modules.

- **PeerTrack Location-Based Storage Service (PT-LBS2).** PT-LBS2 is a PaaS module. It is built on the top of other cloud providers' infrastructure services, such as Amazon EC2, to provide the storage service for RFID reading data or other traceability structures.

- **PeerTrack Index Service (PT-IS).** PT-IS is also a PaaS module. It is a key-value store and responsible for the indexing of various RFID data, such as the latest locations of objects and their moving paths. The indices are stored via the PT-LBS2 to the underlying infrastructures. We do not require the key to be an EPC code. In contrast, they can be anything reasonable.

- **PeerTrack Capture Service (PT-CS).** PT-CS is a SaaS module. Its sole responsibility is to capture the RFID data. It accepts the raw readings and cleans them before sending to the PT-IS for index building and PT-LBS2 for storage. PT-CS complies to the EPCglobal Capture Interface so that the users can configure their readers to send data directly to it.

- **PeerTrack Track and Trace Service (PT-T2S).** PT-T2S is the key module for processing item-level tracking and tracing queries. It relies on the underlying PT-IS. It is also a SaaS module. Note that PT-IS is not only used by PT-T2S. It can be used by any other RFID applications which require quick access to each individual object.

- **PeerTrack Statistical Service (PT-S2).** PT-S2 is the key module for processing statistical tracking and tracing queries. It also relies on the underlying PT-IS. However, the statistical information are stored in main memories of PT-S2 servers and PT-IS is used as the permanent storage. PT-S2 does not invoke PT-IS frequently.

The modularization makes the system flexible so that it complies to the cloud computing philosophy. Users can choose when to join and leave, what services to use and how much to use. For example, if a customer only wants to find an online storage service for his RFID data, he can choose to rent the PT-CS without the PT-IS and PT-T2S. In this case, the data will be sent from PT-CS directly to PT-LBS2. Contrarily, if he wants to be able to track every object, he can choose to rent the PT-T2S and PT-CS. All he needs to do is to configure the readers to send the data directly to PT-CS. Then he can use PT-T2S to do tracking and tracing.

However, there is an exception that the PT-S2 depends on PT-T2S. This is because the statistical information will not be available if the item level information is not acquired, which is done in PT-T2S.

The software services layer (PT-CS, PT-T2S and PT-S2) has no knowledge about how the underlying layers work. They do not need to worry about the replication of data and load balancing. Meanwhile, the PT-IS and PT-LBS2 are responsible for the infrastructure level requirements.

One of the most important features of PeerTrack Cloud is its replication strategy. Unlike other cloud services which choose locations to replicate the data evenly distributed across the globe, we choose to replicate the data at the locations which are close to the related moving path. This is because that in most cases, the data will not be frequently accessed from a non-related location. For example, in a supply chain network, a retailer of a product may frequently access the data at the manufacturer to get information about the product. But it is rarely true that another retailer which is not selling the product (i.e., there is no RFID readings about this product), accesses the data frequently. In fact, in the latter case, the retailer should not be allowed to access that information

at the manufacturer.

Also, data for the objects along the same moving path are mostly stored in the same data center. The benefit is that we do not need to visit more than one data center to answer a tracking or tracing query because the data are stored in the same location. The details about this strategy and the algorithms will be introduced in Section 5.3.

It should be noted that the terminology is a little different in this chapter than that in previous chapters. In cloud computing environments, we do not have the concept of "Node", which represents a location in traceable networks and a server running at that location in previous chapters. Instead, cloud computing environments represent the whole system as a single central server to the users. As a result, we no longer use the term "Node", instead, when we need to refer to a location in the network, we simply use "Location".

5.2.2 PT-T2S Data Model

The PT-T2S maintains the data models used in the tracking and tracing applications and provides interfaces for the applications to query about the location and history of individual objects. In this chapter, we reuse the idea introduced in Chapter 3 that the location of an object is indexed at a gateway. The difference is that PT-T2S does not choose a node as the gateway. Instead, it uses the PT-IS which is the index service handling all the indexing requests. And the underlying storage makes sure that the index is stored successfully and replicated wisely. The index structure is a little different too. Instead of indexing the latest location only (see Chapter 3), we index the full trace for an object. The structure of the index is:

Algorithm : Updating Index in PT-T2S

Input: An RFID reading (o, v, t) from PT-CS
Output: None

```
1: p ← PT-IS.get(o) // get the moving path of o
2: if p is nil
3:    p ← a new list with one record (v, t, t, nil, nil)
4: else
5:    r ← p[p.length − 1] // get the last record in the moving path
6:    if r.location == v // this is a reading at the same location
7:       r.end ← t
8:    else
9:       r′ ← new record (v, t, t, r.location, nil), p.add(r′)
10:      r.to ← v
11:   end if
12: end if
13: PT-IS.put(o, p)
14: if PT-S2 is enabled
15:   PT-S2.put(p)
16: end if
```

Figure 5.4: Algorithm to Update Index in PT-T2S

$$EPC \rightarrow list\ of\ (location,\ start,\ end,\ from,\ to) \tag{5.1}$$

The key of the index is the EPC code, i.e., the ID of an object. The value is a list of records sorted by the *start* field. This list represents the moving path of the object. In essence, we take the distributed linked list introduced in Chapter 3 and make it a non-distributed data structure. The benefits of doing so is twofold. Firstly, in the cloud computing environment, we no longer need to worry about the workload balancing in the software layer (Recall that one of the purposes of distributing the data in Chapter 3 is for load balancing). Secondly, with this simple yet powerful data structure, tracking and tracing queries can be done by a single query to the index, instead of querying the whole network in a P2P way. As a result, the query processing complexity is reduced significantly from $O(log_2|\mathcal{V}|)$ to $O(1)$ in the sense of remote data access.

The algorithm to update the index is shown in Figure 5.4. If the object is first

observed in the whole network, a new record is inserted to its moving path (line 3). Otherwise, if this reading is from the same location as the previous reading, only the *end* attribute is updated (line 6 and 7). If this reading is from a new location, a new record is inserted into the moving path, with the $from$ attribute set as the *location* attribute of the previous record (line 9), and the *to* attribute of the previous record is updated as the new *location* (line 10). Finally, the moving path is put back to the PT-IS service. And if PT-S2 is enabled, this information is also sent to it to update the statistical information.

Compared with the distributed algorithm in Chapter 3, this algorithm is almost the same except that the moving path is updated in the PT-T2S and the only remote calls are the retrieval and update of the index in PT-IS. It is clear that the complexity in terms of remote calls is $O(1)$.

Similar to the distributed algorithm, the algorithm in Figure 5.4 also suffers from the problem that when the volume of RFID streams is high, we have to frequently access the PT-IS. This is a time consuming task and it burdens the PT-IS unnecessarily. To avoid this, we introduce a group-based index update algorithm which is similar but much simpler than the distributed one.

This algorithm is shown in Figure 5.5. The main enhancement in this algorithm is to access the PT-IS in groups (line 2 and line 16). The moving path update part (line 4–15) is exactly the same as the one in Figure 5.4. The underlying PT-IS is responsible for the load balancing and replication of the batches of data. Note that the parameter **T** is introduced as the width of sliding windows for data caching. It is configurable and should be set according to the real-time constraints of the system. Specifically, suppose the maximum allowed delay of index update is **D** and the latency of accessing PT-IS is $\mathbf{T}_{latency}$, we have:

Algorithm : Updating Index in PT-T2S in Groups

Input: RFID readings $\mathcal{R}$ from PT-CS during time **T**
Output: None

```
1: O ← set of IDs of all objects in R
2: P ← PT-IS.getAll(O) // get the moving path of for all objects in O
   // P is a map, whose key is the ID of an object and value is its moving path
3: for each record (o, v, t) in G
4:    p ← P.get(o) // get the moving path of o
5:    if p is nil
6:       p ← a new list with one record (v, t, t, nil, nil)
7:    else
8:       r ← p[p.length − 1] // get the last record in the moving path
9:       if r.location == v // this is a reading at the same location
10:         r.end ← t
11:      else
12:         r' ← new record (v, t, t, r.location, nil), p.add(r')
13:         r.to ← v
14:      end if
15:   end if
16:   P.put(o, p)
17: end for
18: PT-IS.putAll(P)
19: if PT-S2 is enabled
20:   PT-S2.putAll(P)
21: end if
```

Figure 5.5: Algorithm to Update Index in PT-T2S

$$\mathbf{T} \leq \mathbf{D} + 2 * \mathbf{T}_{latency} \tag{5.2}$$

The item-level tracking and tracing query processing algorithm is straightforward. For a given object o, all its information can be retrieved from the PT-IS via a PT-IS.$get(o)$ call and the complexity is $O(1)$. The latest location is the last record in the returned list, and the list itself is the full moving path with time information.

5.2.3 PT-S2 Data Model

In order to gather and represent the changes of object moving patterns in both time and space dimensions, as we do in Chapter 4, we introduce a graph model

and incremental algorithms to maintain the statistical information and capture the dynamicity.

The essential data structure is a directed graph as shown in Figure 5.6. The vertices represent the locations in the network, and the directed edges represent the amount of objects moved along the connections between locations. It should be noted that as shown in the figure, the graph is not necessarily a fully-connected one. The two isolated sub-graphs represent two isolated sub-networks in the system.

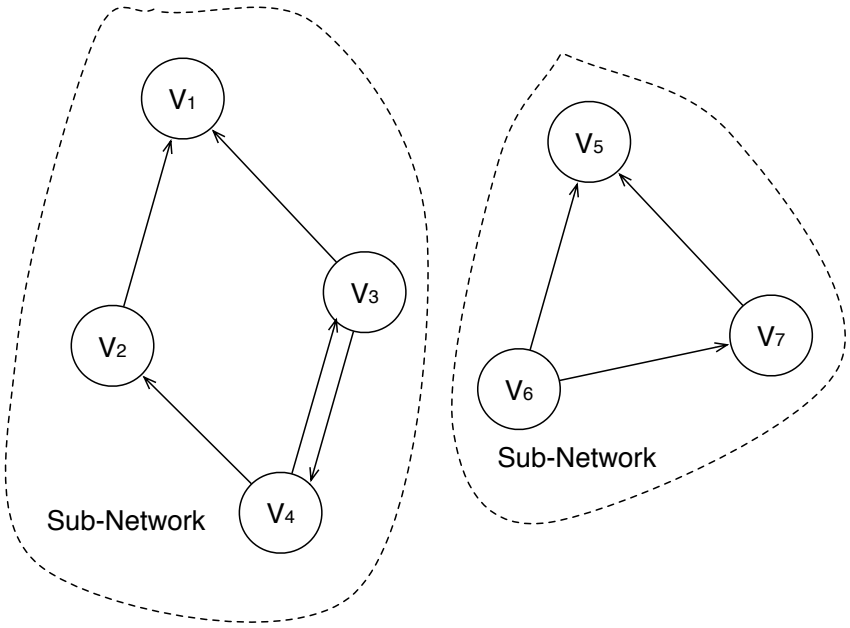
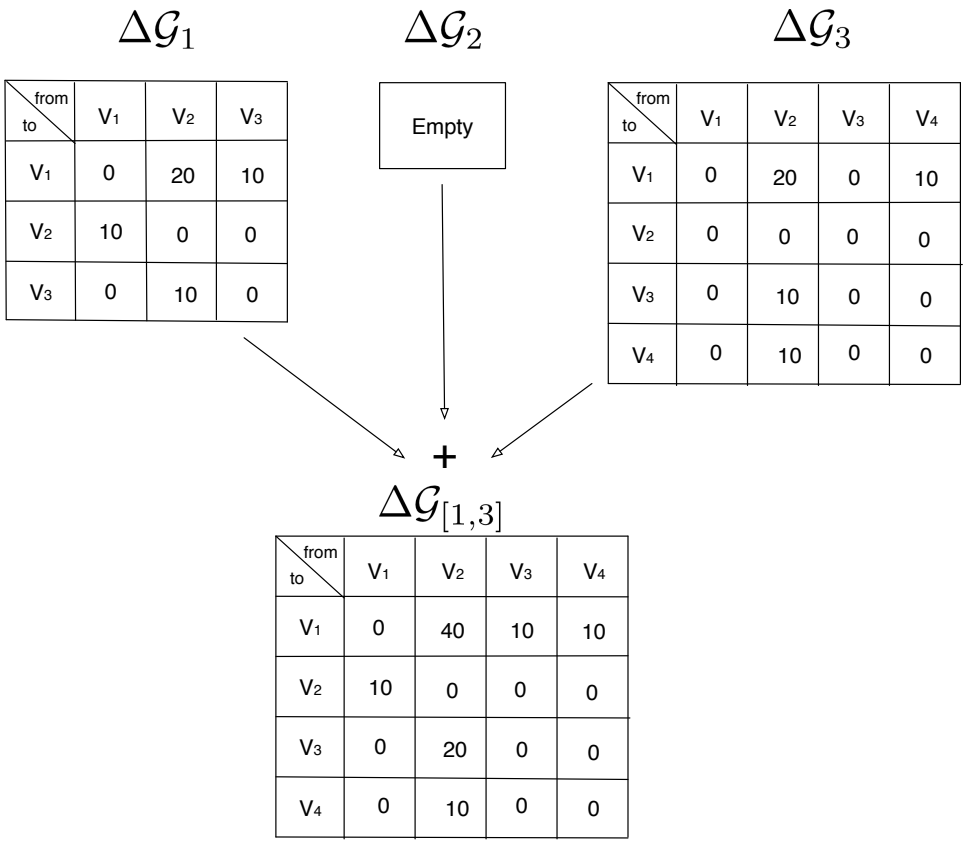


Figure 5.6: PT-S2 Graph Model

In order to capture the changes of the moving patterns, the model is designed to be incrementally maintained. The basic idea is to store the change $\Delta\mathcal{G}_i$ during the i_{th} event cycle. $\Delta\mathcal{G}_i$ is also a graph. It represents the movements happened during the i_{th} event cycle. The width of the event cycle is a configurable parameter **C** which controls the granularity of the statistics. For the convenience of discussions, we name $\Delta\mathcal{G}_i$ the i_{th} *slice*. All the slices are stored in the PT-LBS2

by the PT-S2. Meanwhile, PT-S2 stores the latest **S** slices. The reason is that it is most likely the latest statistics are going to be accessed frequently. In this case, PT-S2 acts as a cache for PT-LBS2. **S** is also a configurable parameter, which balances the storage used in PT-S2 and possibility of answering a query without querying PT-LBS2.



from / to	V_1	V_2	V_3
V_1	0	20	10
V_2	10	0	0
V_3	0	10	0

from / to	V_1	V_2	V_3	V_4
V_1	0	20	0	10
V_2	0	0	0	0
V_3	0	10	0	0
V_4	0	10	0	0

from / to	V_1	V_2	V_3	V_4
V_1	0	40	10	10
V_2	10	0	0	0
V_3	0	20	0	0
V_4	0	10	0	0

Figure 5.7: Example of PT-S2 Graph Model

Figure 5.7 demonstrates the idea using a fully connected network with 4 locations. From the slice $\Delta\mathcal{G}_1$, we can infer that during the first event cycle, there are 20 objects moved from v_2 to v_1 and 10 objects moved from v_1 to v_2. Note that the graph is directed, thus there may be movements in both directions. During the second event cycle, there are no movements at all, thus $\Delta\mathcal{G}_2$ is empty. During the third event cycle, a new location v_4 joined the network and there were 10

objects moved from v_4 to v_1. From these slices, we can infer the statistics for any range. For example, as shown in the figure, the statistics of time range $[1, 3]$ is inferred by combining the three slices together. Similarly, $\Delta\mathcal{G}_{[1,2]} = \Delta\mathcal{G}_1 + \Delta\mathcal{G}_2$ and $\Delta\mathcal{G}_{[2,3]} = \Delta\mathcal{G}_2 + \Delta\mathcal{G}_3$. In general:

$$\Delta\mathcal{G}_{[a,b]} = \sum_{i \leftarrow a}^{b} \Delta\mathcal{G}_i \tag{5.3}$$

Using Equation 5.3, we can retrieve the aggregation information for any time frame.

The algorithm to build $\Delta\mathcal{G}_i$ is straightforward. After the movement data is received from PT-T2S, the changes of locations are transformed to the edges in a slice. The PT-T2S is supposed to send only the records with new locations to PT-S2[5].

5.3 Data Partition and Replication

Data partition and replication are very important topics of data management in cloud computing environments. Their purposes are to increase the availability of data and decrease the access time by directing requests to the nearest data center to the requester. Partition and replication are symbiotic. The whole data space is partitioned to groups according to a certain strategy, then each group is replicated at several data centers. A request is redirected by a high-speed proxy server to the most proper data center based on the network latency and the load balancing policy.

Existing works ([BCD11, XCZ11, CCG11]) are mostly generic solutions and adaptable in most applications. However, RFID data, as one of the location-

[5]This is omitted in Figure 5.5.

based data, has its special characteristics. The most important one is that the distribution of data has certain patterns in the spatio dimensions. For example, in a supply chain network, the supplier-consumer relationships are not changed frequently due to the fact that a contract is often valid for several months or years. Also, a query regarding a path is more likely from a location on the path than from a random location in the whole network, for example, the downstream operations often require information from upstream nodes in supply chain networks. Consequently, if the data is copied to a data center close to the requesting location, the time for data accessing will be shortened. In this section, we introduce the location-based data partition and replication algorithms in PT-LBS2 for traceable RFID networks in the cloud.

It should be noted that the PT-LBS2 is built as a generic service providing storage for all kinds of location-based data. It is not specifically designed for the services in the upper level services in Figure 5.3.

According to whether a record has a key or not and whether the key is a location, we categorize them to three categories: *Non-Location-Keyed*, *Location-Keyed* and *Keyless*. For non-location-keyed records, the same partition-replication schemes are adopted because the requester for those records has no location information, thus the access cannot be accelerated by the location-based replication scheme.

For the other two kinds of records, there are three main location related data structures in the system, namely point, path and graph. Records of these structures can be either key or value in the Location-Keyed records. For these three structures, we introduce different partition and replication algorithms according to their characteristics. In general they all belong to *eager primary copy replication scheme*. I.e., the writing operation is first performed at a primary master

copy[6] and then propagated from this master copy to the secondary copies. The details of this algorithm and other alternatives can be found in [WPS00]. The primary/secondary copies are chosen differently for different structures. Next, we will introduce the methods to determine the copies and to partition the data space.

5.3.1 Point-Based Data Partition and Replication

Point-based data are the records where there is only one location related. The location must be the key if the record is location-keyed, otherwise the record is keyless. For this kind of data, we partition the data space by the location v. Two records $r_1(v_1)$ and $r_2(v_2)$ belong to the same group, if and only if their *closest data center* is the same. The *closeness* is defined by the network latency between a location and a data center[7]. We denote *closest data center* for a location v_i by $cdc(v_i)$.

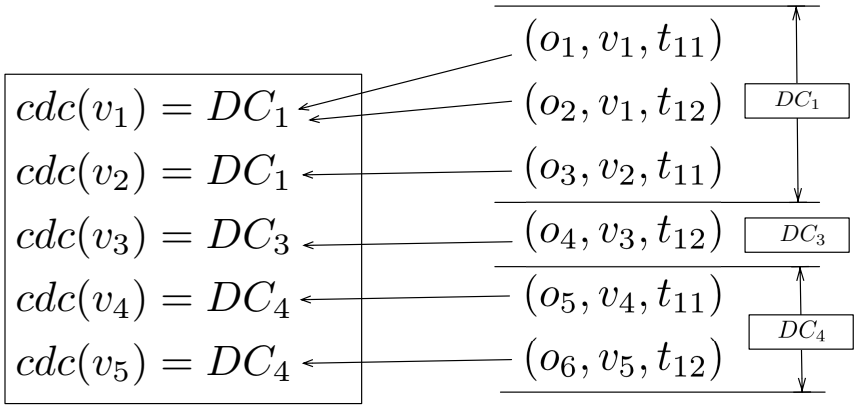


Figure 5.8: Example of Partitioning Point-based Data

Figure 5.8 shows an example of the partition. According to the mapping of the

[6]The request is not always directed to the "Primary master copy". This copy is meaningful only for the update and propagation phase.

[7]To ensure consistency, the network latency is measured once before the system starts. As a result, it is a static measurement and will not change after the system starts.

location to its nearest data center, we partition the 6 records into three groups. A group is identified by the common nearest data center of all its members. The primary copy is stored in the common nearest data center. For a replication policy with m copies, the $m-1$ secondary copies are stored in the order of *closeness* to the primary data center. In this way, the data is stored close to the locations from where it might be requested.

5.3.2 Path-Based and Graph-based Data Partition and Replication

Path-based data are the records where there is an ordered list of locations. For example, the structure we used in PT-T2S to represent a moving path (See Section 5.2.2) is a path-based data structure. Graph-based data are the records where there is a graph of locations. For example, the structure we used in PT-T2S to represent a network (See Section 5.2.3) is a graph-based data structure. Two records $r_1(p_1)$ and $r_2(p_2)$ (p_1 and p_2 are paths) or $r_1(g_1)$ and $r_2(g_2)$ (g_1 and g_2 are graphs) belong to the same group, if and only if they share the common *closest sets of data centers*, which is denoted by $\mathcal{CSP}$ or $\mathcal{CSG}$, for path-based or graph-based data, respectively. $\mathcal{CSP}$ and $\mathcal{CSG}$ are formally defined as ($V(p)$ is the set of locations in p, $V'(g)$ is the set of disjoint subgraphs in g):

$$\mathcal{CSP}(p) = \{cdc(v_i) | v_i \in V(p)\} \tag{5.4}$$

$$\mathcal{CSG}(g) = \{\mathcal{CSP}(g_i) | g_i \in V'(g)\} \tag{5.5}$$

The groups are identified by $\mathcal{CSP}$(or $\mathcal{CSG}$). Figure 5.9 illustrates the scheme by an example of path-based data. For p_2 and p_3, although p_2 and p_3 contain different sets of locations, i.e., $V(p_2) = \{v_1, v_3, v_4\}$ and $V(p_3) = \{v_2, v_3, v_5\}$, since

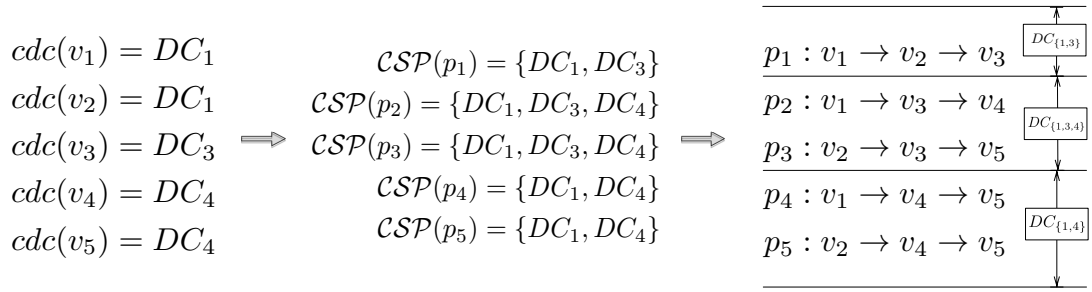


Figure 5.9: Example of Partitioning Path-based Data

v_4 and v_5 share the same closest data center and v_1 and v_2 share the same closest data center too, the two paths share the same $\mathcal{CSP}$. As a result, they belong to the same group identified by $DC_{\{1,3,4\}}$. It should be noted that $\mathcal{CSP}$ is a set, thus the elements within it are not ordered (e.g., p_2 and p_3).

The partition schemes for the two types of data are almost the same. Essentially, the path-based data space and graph-based data space are partitioned by the subsets of all data centers. However, not all the subsets are used. To make the partition and replication more sensible to the location information, this scheme chooses the subsets for which the locations nearby have objects moving among them. The difference between them is how to generate the subsets. For path-based data, it is simple because a path can only generate one subset. It is more difficult for the graph-based data because we have to traverse the graph to find out all the possible paths. We use depth-first graph traversal algorithm to get all the possible paths.

The replication schemes for the two types of data are different.

Path-based Data. The primary master copy is stored at the data center in the $\mathcal{CSP}$ with the smallest ID. For a m-degree replication, if $m > |\mathcal{CSP}|$, the first $|\mathcal{CSP}| - 1$ copies are stored in the unused $|\mathcal{CSP}| - 1$ data centers in $\mathcal{CSP}$. Then the rest $m - |\mathcal{CSP}|$ copies are stored at other data centers evenly distributed.

Otherwise, all copies are stored in data centers in $\mathcal{CSP}$ in the order of their IDs.

Algorithm : Replicating the Graph-based Data

Input: A Graph $\mathcal{G}$ and a piece of data about this graph
Output: m data centers to replicate the data

1: $\mathcal{V}' \leftarrow$ set of subgraphs of $\mathcal{G}$ // using depth-first traversal
2: $\mathcal{CSG} \leftarrow \Phi$
3: **for** each subgraph g in $\mathcal{V}'$
4: $\quad \mathcal{CSP}(g) \leftarrow \{cdc(v_i)|v_i \in V(g)\}$
5: $\quad$ sort $\mathcal{CSP}(g)$ by the IDs of the data centers
6: $\quad \mathcal{CSG}.add(\mathcal{CSP}(g))$
7: **end for**
8: sort $\mathcal{CSG}$
9: $c \leftarrow 0, l \leftarrow 1$ // c is the count of replications, l is the index of data center used in each loop below
10: $\mathcal{RS} \leftarrow \Phi$ // the result
11: **while** $c \leq m$ AND not all data centers are used
12: $\quad$ **for** each $\mathcal{CSP}$ in $\mathcal{CSG}$
13: $\quad\quad \mathcal{RS}.add(\mathcal{CSP}(l))$
14: $\quad\quad c++$
15: $\quad$ **end for**
16: $\quad l++$
17: **end while**

Figure 5.10: Algorithm to Replicate the Graph-based Data

Graph-based Data. As the definition indicates in Equation 5.5, the $\mathcal{CSG}(g)$ is in fact a set of $\mathcal{CSP}$s. Each $\mathcal{CSP}$ is identified by the smallest ID of the data centers within it. If there is a tie (i.e., two $\mathcal{CSP}$ has the same data center with the smallest ID), the second smallest ID is the tie breaker and we identify the $\mathcal{CSP}$ using the first two IDs, and so on. Then the $\mathcal{CSP}$s are sorted by their IDs. To keep the consistency of replication, the primary master copy is stored at the first data center in the first $\mathcal{CSP}$. For a m-degree replication, the rest $m-1$ copies are stored in the other $m-1$ $\mathcal{CSP}$s' first data centers, in the order of their IDs, if $m <= |\mathcal{CSG}|$. Otherwise, after storing the first $|\mathcal{CSG}|-1$ copies in the $\mathcal{CSP}$s, the second data centers in the $\mathcal{CSP}$s are used in the same order, and so on. The detailed steps of this algorithm is shown in Figure 5.10.

5.4 Performance Analysis and Comparison

The main advantage of the PeerTrack Cloud is the ability of handling the real-time constraints requirement[8]. In this section, we briefly analyze the performance in terms of query response time for different queries in PeerTrack Cloud. We also compare the performance with that of EPCglobal Architecture Framework (EAF) and the general P2P solution that we have introduced in Chapter 3.

Tracking Query. The first query that we want to analyze is the tracking query (Q_{track}): find the latest location of an object o. In PeerTrack Cloud, since this is a non-location-keyed record, the PT-LBS2 replicates it evenly distributed in the whole network around the globe. As a result, the performance at worst is two remote access: to the PT-T2S and the data center found in the PT-LBS2. The response time of this query $t_{track}^{PTCloud} = t_{PT-T2S} + \bar{t}_{DC}$. In EAF, the query is sent to Object Naming Service (ONS), then to the Discovery Service (DS). Note that DS is not implemented in EAF at the time of writing[9]. Consequently, we can only assume that the best case for accessing DS is one remote call and the average case is d calls. The worst case is undetermined. Because the DS is most likely to be implemented in a distributed way, in the worst case, $d > 1$. The response time of the tracking query in EAF is calculated as $t_{track}^{EAF} = t_{ONS} + t_{DS} = t_{ONS} + d * \bar{t}_{remote}$. In the P2P environment, the tracking is done by P2P lookups in a DHT, thus the response time is $t_{track}^{P2P} = log_2|\mathcal{V}| * \bar{t}_{remote}$. Clearly, P2P architecture is the slowest to answer the tracking queries, while PeerTrack Cloud provides the fast response because it chooses the closest data center to get the data.

[8]The other common benefits of adopting cloud computing are also provided by PeerTrack Cloud but we omit the discussions about them in this dissertation.

[9]2011-12-01

Tracing Query. The second query that we want to analyze is the tracing query (Q_{trace}): find all the locations that an object o has visited. Similarly, in PeerTrack Cloud, two remote calls are required: to the PT-T2S and the data center in the PT-LBS2. The response time is $t_{trace}^{PTCloud} = t_{PT-T2S} + \bar{t}_{DC}$). It is the same process for EAF to answer the tracking and tracing query. So the response time for EAF is $t_{trace}^{EAF} = t_{ONS} + t_{DS} = t_{ONS} + d * \bar{t}_{remote}$. In the P2P environment, since the trace is distributed along the moving path, it is required to access all the nodes along the path to get the information. As a result, the number of remote calls is the number of nodes in the resulting path. The response time is $t_{trace}^{P2P} = \bar{l} * \bar{t}_{remote}$ where $\bar{l}$ is the average length of the moving paths. Similar to the analysis for tracking query, PeerTrack Cloud provides the fast response for tracing query too.

Statistics Query. The third query that we want to analyze is the statistical query (Q_{stat}): find the number of objects sent to location v during last month. In PeerTrack Cloud, this is a location-keyed record, so it is replicated to the closest data center to the location in the request and its neighbors. Though we still need two remote calls : to the PT-S2 and the data center in the PT-LBS2, the data center chosen is likely near the requester. The response time is thus $t_{stat}^{PTCloud} = t_{PT-T2S} + min(t_{DC})$. It is hard to tell the performance of the EAF for this kind of queries since this highly depends on the implementation. In the P2P environment, because the data is stored at where it is collected, i.e., data about v is stored at v, the number of remote calls is the number of hops to get to the node. The response time is $t_{stat}^{P2P} = \log_2 |\mathcal{V}| * \bar{t}_{remote}$.

We summarize the comparison of performances in different architecture for these three major kinds of queries in traceable RFID networks in Table 5.1. The actual measurements may depend on the configurations and environments. However, it is evident that PeerTrack Cloud outperform the other two architectures

Architecture	Response Time for Query		
	Tracking	Tracing	Statistical
PeerTrack Cloud	$t_{PT-T2S} + \bar{t}_{DC}$	$t_{PT-T2S} + \bar{t}_{DC}$	$t_{PT-T2S} + min(t_{DC})$
EAF	$t_{ONS} + d * \bar{t}_{remote}$	$t_{ONS} + d * \bar{t}_{remote}$	N/A
P2P	$log_2\|\mathcal{V}\| * \bar{t}_{remote}$	$\bar{l} * \bar{t}_{remote}$	$\log_2 \|\mathcal{V}\| * \bar{t}_{remote}$

Table 5.1: Comparison of Performance in Different Architectures

in most of the scenarios, in terms of response time.

It should be noted that this analysis measures the response time without considering the influence of workload at the servers. In the environment where the volume of the data is high, if the number of servers used in EAF's DS is small, each server gets high load and the response time is downgraded. Otherwise, when there are a lot of servers in DS, the architecture requires more remote accesses. As a result, it downgrades too in terms of response time. So the performance of EAF depends on the environments and the actual design of DS.

5.5 Related Work

The researchers in UC Berkeley summarized the challenges and opportunities of Cloud Computing in [AFG10] and [AFG09]. Among the ten obstacles that they discussed, which include technical, financial and management issues, *Data Transfer Bottlenecks* and *Scalable Storage* are the most interesting topics of data management. There are many opportunities in the research of these areas. Another state-of-art overview of cloud computing [ZCB10] focused on the technical challenges. The authors provided a comprehensive analysis and comparison over the existing cloud products from the biggest providers (Amazon, Google and Microsoft) from the architecture, file storage and management aspects.

Cloud computing for traceable RFID networks is a sub-topic in the big picture.

In this section, we overview the work that have done for mobile and location-aware, especially RFID-based, cloud computing.

In [KMS10], a framework for the use of context information provided by mobile cloud as a service is proposed. This work introduces the Heterogeneous Access Management (HAM) concept and proposes a Context Management Architecture (CMA) to acquire, process and manage the context information. This is a generic framework for context-aware mobile services. In [LZZ09], the authors propose a middleware framework for cloud computing to deploy mobile computation, especially mobile agent technology, in cloud services. Service composition and service invocation techniques in this architecture are also introduced. These architectural works are mainly focused to provide a generic framework for the mobile cloud computing environment. As a result, they do not specifically consider the traceability as a requirement. In [ZFM10], a framework designed for high-speed data access in RFID networks is proposed. The authors present experiments to explore the main bottlenecks in EPCglobal architecture to provide a real-time environment for RFID data processing. The proposed cloud-based service can provide real-time access to EPCIS. However, this work does not address traceability either. To the best of our knowledge, traceability as a service is still in its early stage of development.

Data management is a very important topic in cloud computing. In [PSL11], a locality-aware resource allocation method is propose. It enables quick access to data by reading them from local or close-by physical machines. This work is mainly developed to expedite MapReduce jobs. Thus it is not generic enough to be suitable for enabling traceability. In [CCG11], the authors proposed a storage system for supporting both OLTP and OLAP, similar purpose for the work in this chapter to support both item-level and statistical traceability queries. Though

this work is generic so that it can support traceability queries as well as queries in other specific topics, it uses a P2P index system which causes longer response time. In [XCZ11], the authors try to address the issue of how to intelligently manage the resources in a shared cloud database system. They propose SmartSLA, a cost-aware resource management system which uses machine learning techniques to learn a model that describes the possible profit margins and use it to optimize the resource allocation. This work focuses on the management of resources in terms of costs. We take this topic as a future research direction.

5.6 Summary

In this chapter, we introduced PeerTrack Cloud, which is a cloud computing architecture built to enable Traceability as a Service (TaaS). The main features of this system include:

- We modularize the architecture in layers, in order to provide a flexible and affordable ecosystem for users with various needs. Users do not need to sign up for all the features if they only want a specific one.
- The architecture features two major SaaS layer modules, PT-T2S and PT-S2, for item-level traceability query processing and statistical query processing, respectively. We proposed algorithms for the query processing and data management in these modules.
- One of the purposes of this architecture is to enable real-time access to various data, to which the key is how to access the data as quickly as possible. We designed a storage service (PT-LBS2) specifically for traceability data structures, such as location, path and graph. Algorithms to partition and replicate the data are designed to further expedite the data access.

The details about the implementation and performance evaluation can be found in Chapter 6.

Chapter 6

Implementation and Performance Study

This chapter is devoted to the implementation and performance study of our proposed solutions for traceability. We implemented these techniques inside the PeerTrack Platform. PeerTrack aims at providing a comprehensive platform for enabling traceability in large-scale, distributed RFID networks. To validate the feasibility and benefits of our approaches, we developed an asset management system (PeerTrack AMS) using this platform, which provides support for various kinds of traceability queries. We then conduct an extensive performance study of our approaches.

This chapter is organized as follows. In Section 6.1, we give a brief overview of the PeerTrack platform. In Section 6.2, we describe some implementation details of PeerTrack. Then, in Section 6.3, we present the PeerTrack AMS that illustrates the main features of PeerTrack. In Section 6.4, we reports the results of a set of performance studies. Finally, in Section 6.5, we provide a summary of this chapter.

6.1 PeerTrack Platform: An Overview

Enabling traceability is not a single layer problem [WRS11]. Large-scale global networks have the potential to generate unprecedented amounts of data related to individual objects. An important challenge centers on the efficient management and sharing of this data in traceability applications. An obvious solution is to publish all data collected within each organization to a central data warehouse. Unfortunately, this approach has several severe drawbacks. Firstly, object movement and related data are valuable business information that companies may be very reluctant to put in a shared central warehouse. Secondly, such an approach has very limited scalability and is not feasible for large-scale applications where the amount of data collected could be enormous [ACK06b, FJK05, SLZ08a, WJO09]. The system architecture for data gathering, processing and sharing must be scalable in order to deal with the data collected from networked systems. For efficient processing and storage, data models must be carefully designed. To allow business users making useful decisions and analysis in a timely manner, different types of traceability queries as well as event-driven notification services must be conveniently supported.

Motivated by these concerns, we have developed the PeerTrack platform for efficiently tracking and tracing objects in large-scale traceability networks. PeerTrack features a pure P2P architecture for data and query processing. In particular, we have implemented the data models introduced in Chapter 3 and Chapter 4, which eliminate the data dependencies between organizations. Important tracking and tracing queries have been implemented as built-in features, meanwhile we provide the flexibility of developing add-in queries.

We present the architecture of the PeerTrack Platform in Figure 6.1.

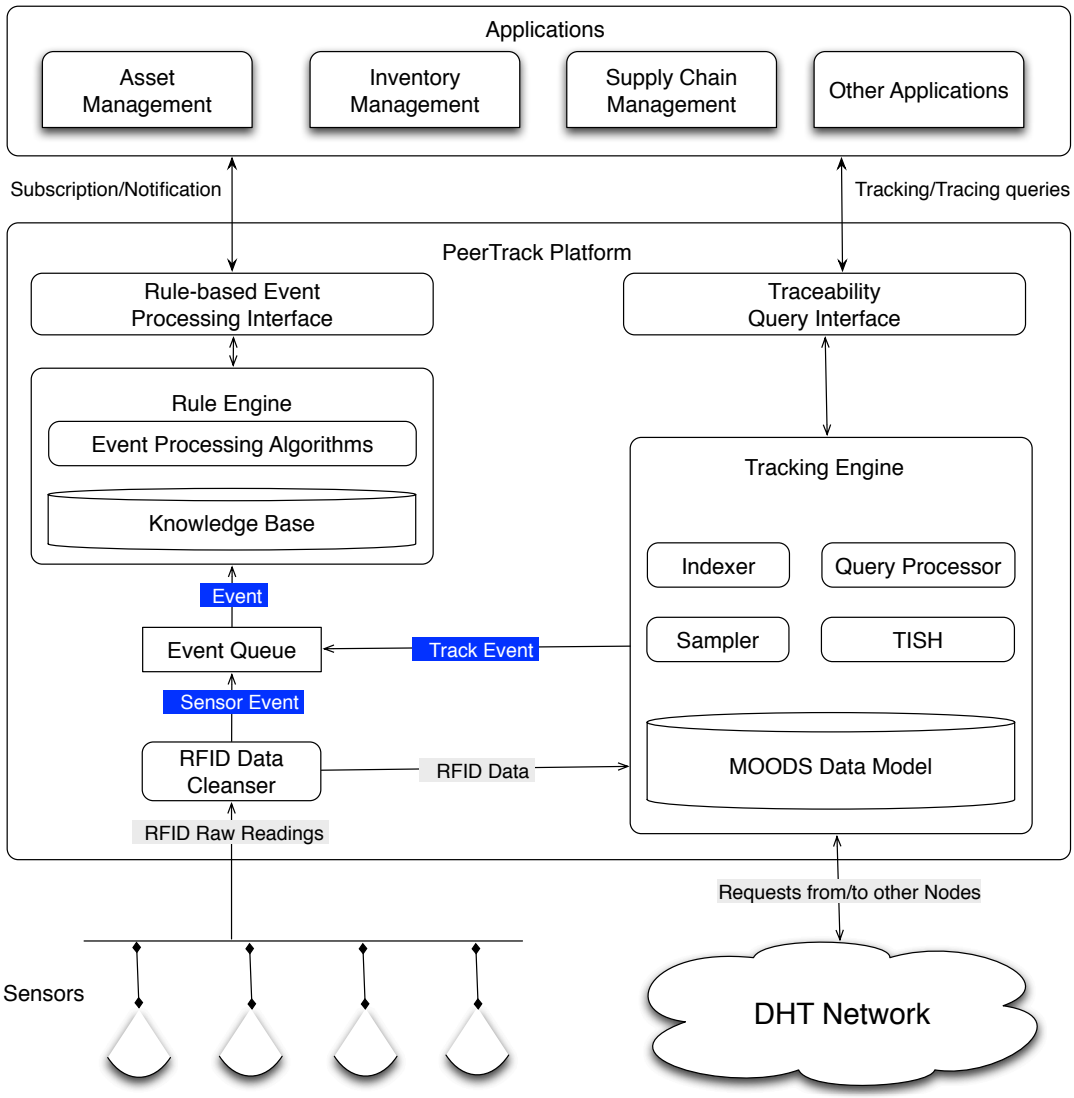


Figure 6.1: The Architecture of PeerTrack Platform

PeerTrack exploits the distributed hash table (DHT) infrastructure Chord [Sto01] to manage peers and route messages. Each organization is viewed as an equal peer of the network. "PeerTrack Platform" in the Figure 6.1 shows the structure of a peer. Within a peer, applications can access local data, as well as remote data of other peers using the *tracking engine* via the DHT network.

The tracking engine includes traceability data models (MOODS and TISH), an indexer, and a distributed query processor. Developers can implement various kinds of traceability applications using the generic APIs built on top of these modules. To support event-driven services (e.g., notifications), a rule engine has been developed. It monitors an event queue which is public to other modules.

PeerTrack relies on the generic data model MOODS(See Chapter 3) for tracking and tracing each object individually in large-scale networks. A discrete space refers to a finite set of nodes which represents all the organizations in the network. MOODS eliminates the data dependencies between organizations by storing the information about object movements at the nodes where the object has been transported. In particular, MOODS introduces the distributed double linked list to represent the path information, which includes properties that indicate the departure and arriving information of objects. With this data structure, each node maintains segments of objects' moving paths and uses this information to expedite P2P queries in the network. Details about MOODS can be found in Chapter 3. TISH is implemented to provide mining support. It gathers the information from peers without having to use large amount of bandwidth. TISH can run in the main memory because its small memory footprint. However, it is powerful enough to support mining over a long period of time. Details about TISH can be found in Chapter 4.

The indexer is the key to maintain the distributed moving paths. It indexes

an observed object and its latest location (i.e., the last node where the object is observed) at a deterministic node called the *gateway node*. The gateway node is solely determined by the id of the object thanks to the determinism of the DHT. When the object is observed at a new node v_d, the node sends the object's id to its gateway node via the indexer. The indexer at the gateway node uses this information to update its index and sends a message back to v_d, notifying it the node where the object comes from (i.e., v_s). Meanwhile, the indexer also sends a message to v_s, informing the node that the object has arrived at v_d. As a result, the moving path is established. To reduce the indexing overhead for large-volume objects, we enhance the indexing algorithm by grouping the objects according to the prefixes of their hashed ids. The scheme to determine the optimal length of prefixes for grouping is carefully chosen in regard to both scalability and load balancing.

The fundamental design principle of the query processor is to process a query locally to the extent possible and, if necessary, enhance it using locally available information before forwarding it to appropriate remote organizations. Due to the introduction of MOODS and TISH, we do not have to flood the query to all the nodes in the network. Instead, the query is processed following the distributed link. As a result, the performance of query processing can be significantly improved. To answer a tracking query, i.e., finding the current location of an object, the query processor simply contacts the gateway node for the object via Chord. A tracing query, i.e., finding the pedigree of an object, can be answered by first tracking the object (i.e., finding its current location) and then simply tracing back the list using the moving path information.

We used state-of-the-art technologies for the implementation. Table 6.1 summarizes the technologies that we have used. The two interfaces, *Traceability*

Product	Version	Usage Descriptions
MySQL	5.5	DBMS Server.
xerces	2.4	XML processor.
JBoss Application Server	5.0	Java EE Application Server.
Drools	5.0	Event-Rule Processor.
iText	5.0	document generator.
log4j	1.2.15	logger.
MySQL JDBC Driver	5.1.16	Database connections.
Fosstrak ALE Middleware	1.0.2	ALE implementation.
Axis	1.4.0	SOAP provider.
OpenChord	1.0.5	DHT implementation.

Table 6.1: Enabling Technologies in PeerTrack Platform

Query Interface and *Rule-based Event Processing Interface* are deployed as web services using Apache Axis. JBoss is used as our web and application server where Axis is deployed. The *Rule-based Event Processing Interface* contains the essential methods provided by the *Rule Engine*. We implemented the *Rule Engine* using the open source event-rule processor, Drools[1]. Drools contain 5 modules, of which we used Drools Expert and Drools Fusion as our rule and event processing modules, respectively.

We adopted the open source Java project OpenChord[2] as the implementation of Chord [Sto01]. It supports storing all serializable Java objects within the DHT.

6.2 Implementation Details

The PeerTrack platform provides an environment for supporting the development of distributed and large-scale traceability applications. It offers two interfaces to application developers, namely the *traceability query interface* and the *rule-based event processing interface*[3], in the form of Java API. The query interface accepts

[1] `http://www.jboss.org/drools`

[2] http://open-chord.sourceforge.net/

[3] For convenience of discussion, we will refer to them as "query interface" and "rule interface".

queries and executes them either locally and/or remotely at other nodes. The rule interface can be used to specify business rules for event-driven services (e.g., out-of-stock alerts). These two interfaces are built on top of PeerTrack's main modules, namely the *tracking engine* and the *rule engine*.

6.2.1 Rule Engine

Listing 6.1: Example of a Rule

```
rule "Order Created"
when
    $order : Order(status == OrderStatus.CREATED)
then
    for (Object o : $order.getObjects()) {
        o.setStatus(ObjectStatus.PROCESSED);
        o.setOrder($order);
        insert(o);

        QueryProcessor.getInstance().getQueryProcessor().
            updateObjectStatus(o);
    }
    QueryProcessor.getInstance().getQueryProcessor().
        createOrder($order);
    FeedGenerator.getInstance().generateOrderCreatedEvent(
        $order);

    System.out.println("order created : " + $order.getId());
end
```

Listing 6.1 shows an example of a rule ("Order Created"), which is triggered by the event (indicated by the keyword "when") that an order is input into the system. The rule engine accepts the definition of rules as the example, via the rule interface. The applications can then subscribe to the defined events, and get notified when they occur. The rule engine is built on top of *JBoss Drools*[4], which includes five main modules. We used Drools Fusion and Drools Expert to build our rule engine. The rule engine monitors an *event queue* which receives events from other modules in PeerTrack and triggers corresponding actions if the conditions are satisfied.

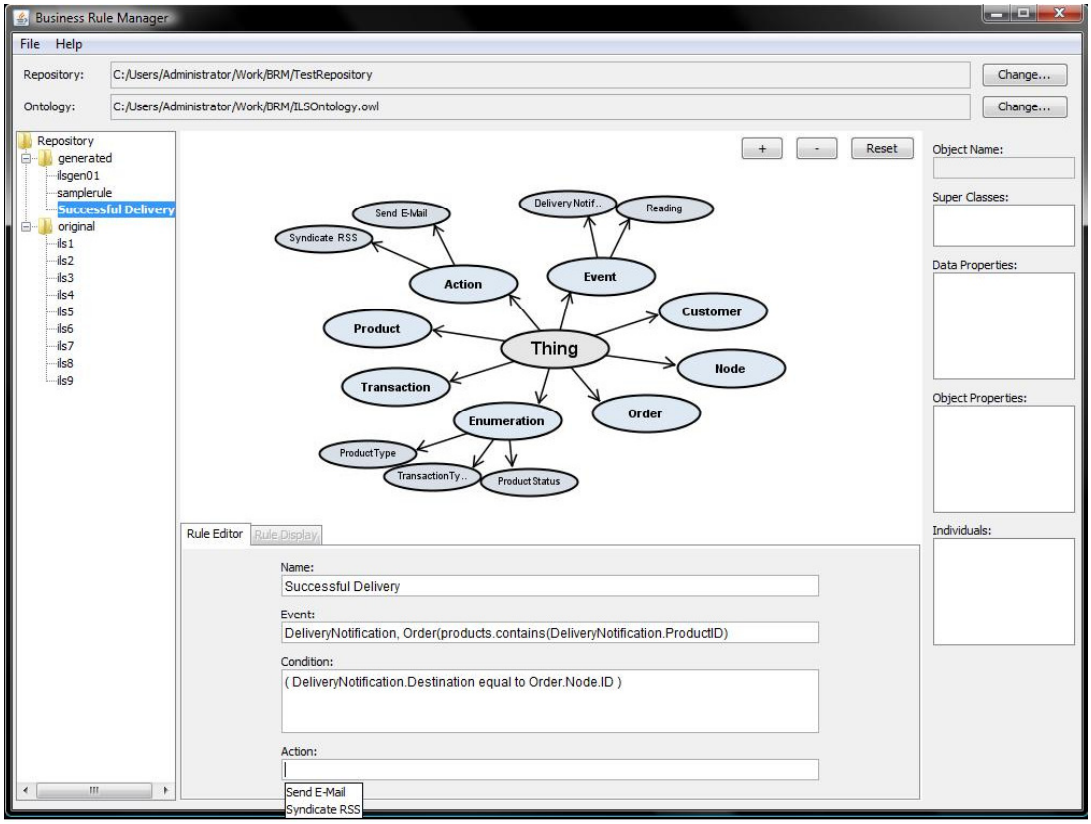


Figure 6.2: Screenshot of the Rule Editor

[4]http://www.jboss.org/drools

The editing of the rules is not simple because it requires programming skills as shown in Listing 6.1. In order to make the system easier to use for people without programming skills, a rule editor (Figure 6.2) is developed to provide a visual interface for efficiently defining and editing business rules. It also contains a knowledge base that assists users who do not have specialized knowledge to define business rules with a semi-structured natural language. The rule editor supports the semantic specification of rules and automatically translates the visual representation of rules to the Drools Rule Language.

The event filterer and cleanser have been implemented to filter and clean the data stream collected from various data sources, e.g., RFID readers and barcode readers. It mainly implements the algorithms proposed in [JAF06].

6.2.2 Tracking Engine

Tracking engine contains several major modules, namely *Indexer*, *Query Processor*, *Sampler* and *TISH*.

The indexer is transparent to the application developers. It implements the group indexing and model maintenance algorithm described in Chapter 3. The indexer reads data from the stream processor periodically. Objects observed within a cycle are classified into groups according to the prefixes of their hashed ids. We use SHA-1 as the hash function for its uniformity. The length of the prefixes is determined by the function $\log_2 |\mathcal{V}| + \log_2 \log_2 |\mathcal{V}|$, where $|\mathcal{V}|$ is the size of the network, represented by the number of nodes within the network. The anti-entropy aggregation protocol proposed in [JM04] has been implemented to estimate the value of $|\mathcal{V}|$.

The query processor accepts queries from both the application layer and the DHT layer (rewritten from other nodes). A `Dispatcher` class has been im-

plemented to pick up a corresponding registered processor instance for an incoming query. All queries and their processor instances are implemented as plug-ins. For example, for a tracking query `TrackQuery` that locates an object, a `TrackQueryProcessor` is implemented. Both classes are registered in the query processor via the `register(Query` *q*`, Processor` *p*`)` interface at runtime, where `Query` and `Processor` are the Java classes for all queries and processors. This mechanism ensures high flexibility and makes it possible to dynamically upgrade the system without rebooting. We have implemented some important traceability queries. A special implementation is the `RewritableQuery` class, which represents queries to be rewritten for executing at other nodes via DHT interface. Tracking and tracing queries are subclasses of the `RewritableQuery`. The query objects are all serializable for seamless integration with OpenChord.

The sampler accepts RFID data from the readers after they are cleaned, then outputs a small set of records which is a true random sample of the input dataset. This is guaranteed by the *Reservoir Algorithm* [Vit85]. The data set is read only once so there is no cache for it. This makes the footprint of the sampler very small.

The TISH module maintains the model that we proposed in Chapter 4 in the main memory. It accepts the sampled dataset from the sampler, and asks other nodes via the DHT interface about the origin of the objects in the sample. Upon the receipt of the results, it updates its Titled Time Frame of Histograms. All these operations are done in the main memory, thus it is quick and efficient. Also, because the dataset has been sampled, the bandwidth usage is saved.

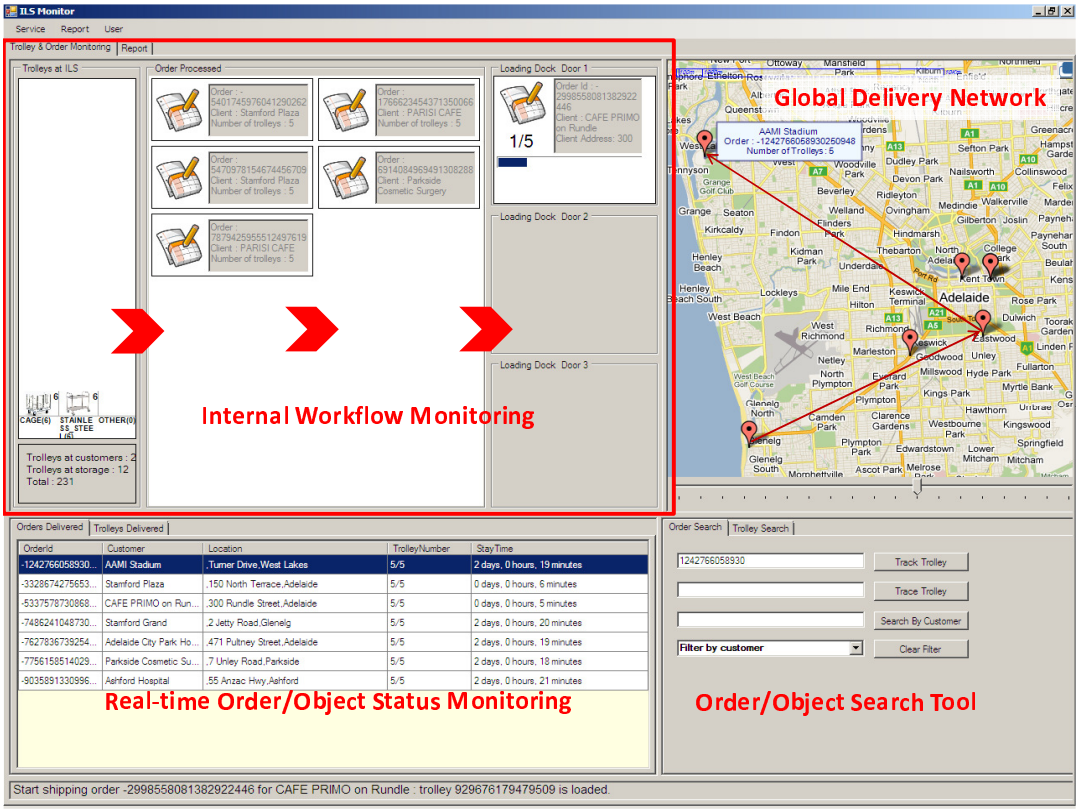


Figure 6.3: Screenshot of the PeerTrack AMS Client

6.3 PeerTrack AMS: A Demonstration

We have developed several traceability applications on top of the PeerTrack platform, including mobile asset management and supply chain management. In this paper, we will focus on presenting a mobile asset management system. This system has been deployed at a company that provides a suite of linen services for over 200 customers (e.g., hotels, hospitals, and aged-care homes) in South Australia. Each customer has one or more delivery locations (nodes). In this system, trolleys (objects) are reusable containers for linens and they are attached with RFID tags. They are transported among nodes and can be detected by RFID readers when they arrive at a delivery location. An order is simply an aggregation

of several trolleys for the same customer. The mobile asset management system developed from the PeerTrack platform offers an automated tracking and tracing service with the capability to monitor and control their logistical operations in real-time over 300 different locations.

We also developed a visual monitoring tool that has been deployed at each customer's site, together with the P2P services. Figure 6.3 shows the screenshot of the visualization tool deployed at the clients' desktop machines. We use this application as the demo to show the advantages of our PeerTrack platform.

Tracing and Tracking. Tracing a trolley can be initiated using the "Order/Object Search Tool" panel. After a user inputs the trolley id and clicks the "Trace Trolley" button, the visualization tool creates `TraceTrolleyQuery`, which is a subclass of the `TraceQuery`, and sends it to the PeerTrack platform. PeerTrack traces it to the original location of the trolley following the moving path stored in each node along its moving path. The search result is shown as highlighted lines on the map of the query initiator.

Similarly, for tracking a trolley, the `TrackTrolleyQuery` is sent to the corresponding gateway node via the underlying structured overlay. The gateway node which maintains its latest location then sends back the node id of the current location of the queried trolley. The result is shown at the query initiator as an informational bubble (see the one at the left top of the "Global Delivery Network" map in Figure 6.3).

With this architecture, the tracking and tracing can be done with minimum number of network calls. In the implementation, to further reduce the network cost, we cache the IP address of customers so that in most cases the underlying P2P routing cost is eliminated.

Inventory monitoring. The query processor offers facilities for developer to

implement various kinds of queries and register them into the system. To realize quasi-real-time inventory monitoring, an `InventoryQuery` class (which is subclass of `RewritableQuery`) and its processor `InventoryQueryProcessor` are implemented. The former defines the query parameters including the node to monitor and the refreshing interval. When the query is issued to the query processor, an `InventoryQueryProcessor` instance is initialized for query processing. Specifically, the `InventoryQueryProcessor` periodically sends the `InventoryQuery` object via OpenChord to the monitored node.

When the node icon is clicked, a context menu is popped up with the option to start monitoring the node. The inventory is displayed near the node (e.g., the number "50", "100") and refreshed at a certain interval.

Real-time Monitoring. The users can define various kinds of rules in PeerTrack and get notified when an event of interest occurs. The rules for successful and incorrect deliveries have been created with the rule editor (Figure 6.2 shows the definition for the "Successful Delivery" rule). When the successful delivery rule is triggered, i.e., a trolley is delivered to the correct destination, the "Real-time Order/Object Status Monitoring" area in Figure 6.3 is updated with the latest information. Meanwhile, the "Global Delivery Network" map is updated by showing an icon at the destination of the delivery.

This is realized by the indexer and the rule engine. As part of the moving path acquisition process, the indexer will be notified by the gateway node after an object has arrived at another node. This notification, as an event, is also sent to the event queue which is monitored by the rule engine. In the application, we also implemented the "Internal Workflow Monitoring" sub-system that monitors the internal movements of the trolleys.

6.4 Performance Study

We conducted experiments for the technologies proposed in Chapter 3 and Chapter 4 using the implemented prototype system, and conduct experiments for the PeerTrack Cloud proposed in Chapter 5 using a simulation tool. This section presents the experimental results for them.

6.4.1 Performance Study on the P2P Architecture and MOODS

In this experiment, we want to verify that the P2P architecture built on the top of MOODS is scalable and efficient. Due to the limit of resources that we have and the requirement of large-scale network, we replaced the real DHT overlay with a simulated Chord implementation in OverSim [BHK07]. In our tests, the maximum number of nodes was 512 and the maximum number of objects at each node was 5,000, corresponding to the limit of our experiment environment. All experiments were conducted on an Intel Core 2 Quad 2.4GHz system with 4GB of RAM.

6.4.1.1 Performance Study on Scalability

We compared it with a centralized solution which we have implemented according to [WL05a]. The comparison was done on four queries which belong to different categories that we summarized in Section 2.2.2. The dataset was generated at random. First, we generated a certain number of objects at each node. Then we kept the system running for 10,000 event cycles. During each event cycle, a set of randomly chosen objects at each node were moved from one node to another randomly chosen node to build the moving paths. Thus, the maximum length of a moving path was 9,999 and the minimum one is 0. Finally, we ran

	Query
Q1	Where has object o_i been?
Q2	Where is object o_i?
Q3	How many objects moved from node v_i to node v_j?
Q4	How many objects moved along the path $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k$ during last month?

Table 6.2: Testing Queries for Scalability of PeerTack and MOODS

the various queries in the network from a randomly chosen node. The results presented in this section are the average values of running the same query with different parameters (i.e., different objects or nodes).

The concrete queries are listed in Table 6.2.

Firstly, we fixed the number of objects at each node as 5,000 to see how the system scales against the size of the network. Figure 6.4 shows the results when we test on networks of size 64, 128, 256 and 512. From the figure we can see that the query processing time increases slowly for the P2P approach, but increases sharply for the centralized one. Since the data at each node in different settings (number of nodes in the network) is of the same amount, the local processing time does not change. Then, the query time of Q1, Q2 and Q4 in P2P network is proportional to the logarithm of the size of the network. For Q3 (Figure 6.4 (c)), the cost for Q3 almost stays constant for P2P approach. This is because that there is no P2P lookup involved. However, the cost for centralized approach increases with the number of nodes.

For the centralized approach, because all the data is stored in the same database, when the size of the network increases, the amount of the data in the central database increases too. The time for querying is relevant (ultra-linear because of the join queries used) to the size of the database, which is proportional to the size of the network when the number of objects generated at each node is

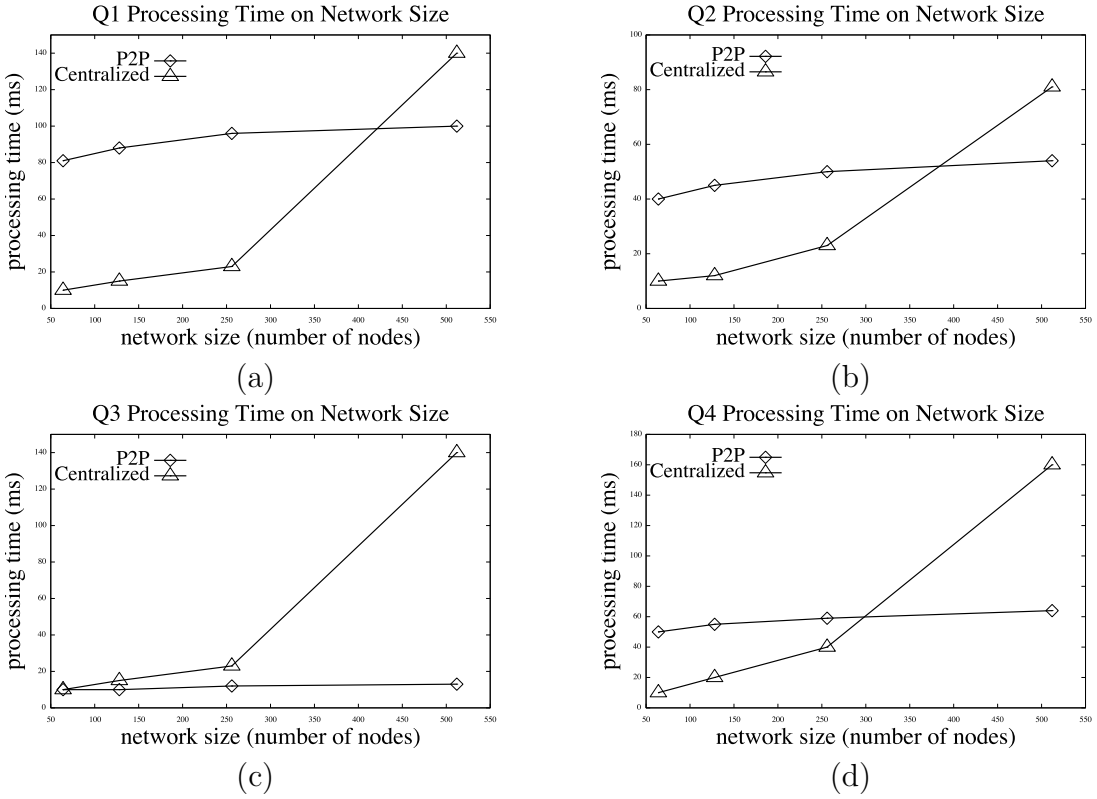


Figure 6.4: Scalability on Network Size

fixed.

We can also see that when there are less nodes, centralized approach outperforms P2P one. However, when the number of nodes and records in the network reaches certain amount, which is typical in large scale RFID networks, it will run more slowly than the P2P approach.

Secondly, we fixed the number of nodes in the network at 512, while ran the same queries with number of objects at each node from 500 to 5000, at a step of 500, to see how the system scales against the volume of data. Figure 6.5 shows the query processing time for different data volumes. Similar result was obtained. The processing time for Q1, Q2 and Q4 increases much more slowly in P2P approach than in centralized one. The processing time for Q3 in P2P

approach increases too (Figure 6.5 (c)). This is because the volume of data at each node increases. The high scalability shown in the experimental results is due to the distributed linked list in our design.

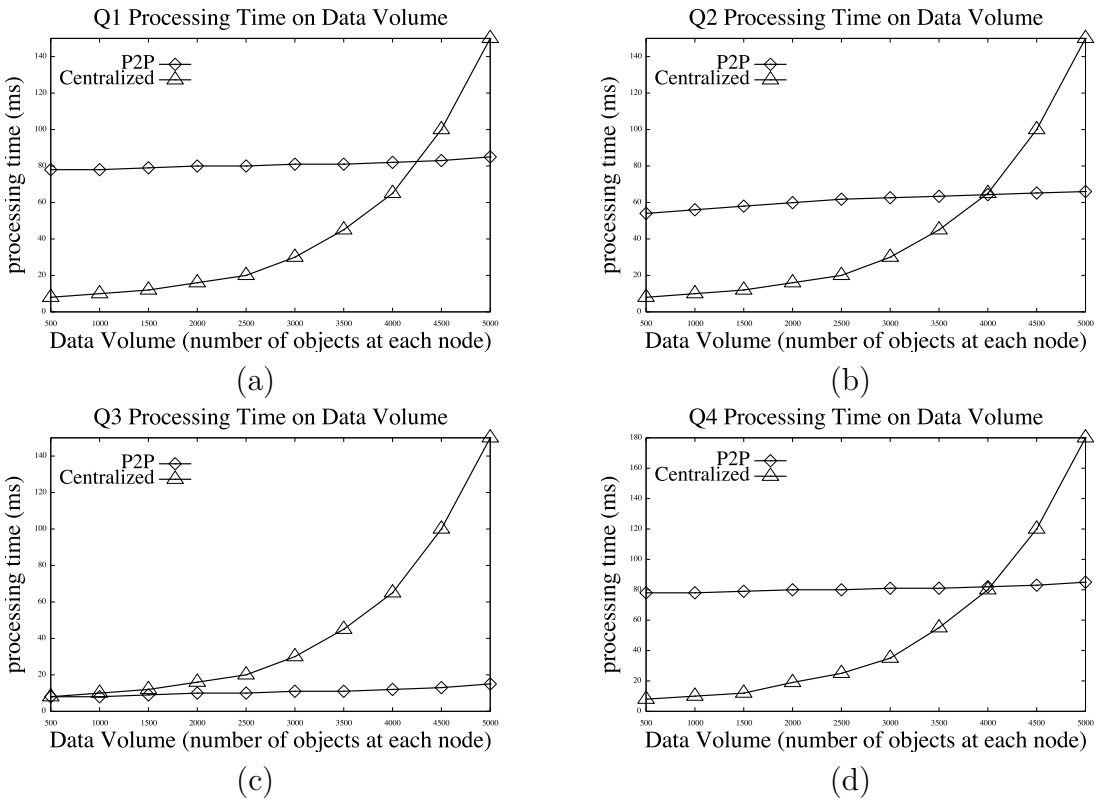


Figure 6.5: Scalability on Data Volume

From the experiments, we can conclude that the P2P architecture and the MOODS data model can realize a highly scalable traceable network.

6.4.1.2 Performance Study on Bandwidth Efficiency

The process of building the index in a P2P fashion introduces a larger usage on bandwidth. We proposed a group-based indexing algorithm in Section 3.3 to save the bandwidth costs. In this experiment, we verify that this algorithm can significantly reduce bandwidth usage.

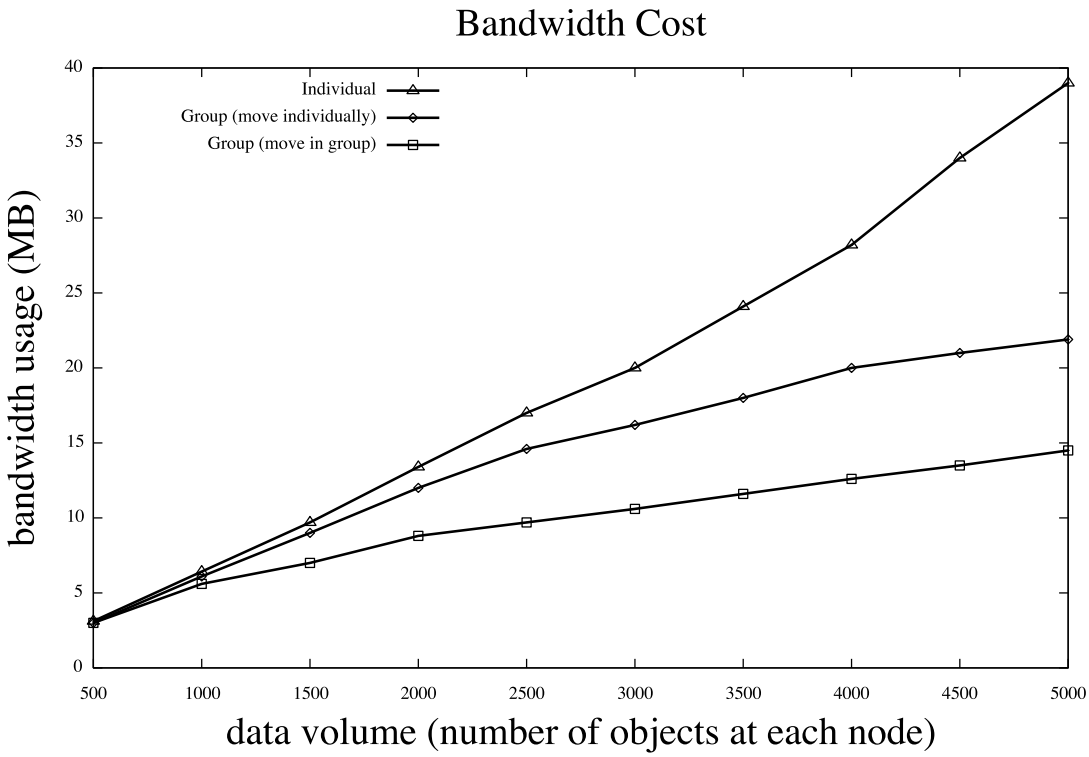


Figure 6.6: Bandwidth Cost in Different Scenarios

In this experiment, we built the dataset as we did in Section 6.4.1.1, and recorded the total size of packets sent from all the nodes in bytes. The number of nodes was fixed at 512. We ran the test 10 times and for the i_{th} ($1 \leq i \leq 10$) time, the number of objects generated at each node is $500*i$. To simulate the movement of objects, 10% of the local objects at each node were moved along a trace of 10 nodes. The process was run on three different settings: 1) indexing the objects individually, 2) indexing the objects in groups, with objects moving individually, and 3) indexing the objects in groups, with objects moving in groups. The result is depicted in Figure 6.6.

From Figure 6.6 we can see that, when the data volume is not high (e.g., 500), the group indexing algorithm does not show much of an advantage. This

is because that when the number of objects is small, most of the groups contain only one or two objects. The number of groups is close to the number of the objects. Thus the group indexing algorithm costs almost the same as the individual indexing algorithm. However, with increasing data volume, the indexing cost of the group indexing algorithm increases much slower than the individual indexing algorithm, because data is grouped and compressed before sent to gateway node. It is also clear that when the objects move in groups, the indexing cost is further reduced. Even when there are fewer objects, the performance is much better. This is because that when objects move in groups, it is highly possible that most objects fall into the same window, so in that window, the number of objects is high. In reality, particularly large-scale applications, there are much more objects than nodes. Clearly, these applications can take advantage of the benefits brought by our proposed group indexing approach.

6.4.1.3 Performance Study on Load Balancing

Load balancing is an other important requirement of P2P applications. In this experiment, we verify that the carefully chosen parameter l (the length of grouping prefix) can guarantee that the workload is well balanced. In this experiment, we studied the different schemes of l and tested their corresponding load balancing capabilities. Figure 6.7 shows the result for the three different schemes (i.e., Scheme 1: l=$\log_2 |\mathcal{V}|$, Scheme 2: l=$\log_2 |\mathcal{V}| + \log_2 \log_2 |\mathcal{V}|$, and Scheme 3: l=$2 \log_2 |\mathcal{V}|$). Scheme 2 is the one that we have chosen for our system (which is also the scheme used in all other experiments). We studied Scheme 1 and Scheme 3 in this experiment as they are asymptotically smaller/bigger than Scheme 2, respectively.

We illustrate the load balance by showing the load percentage (i.e., the num-

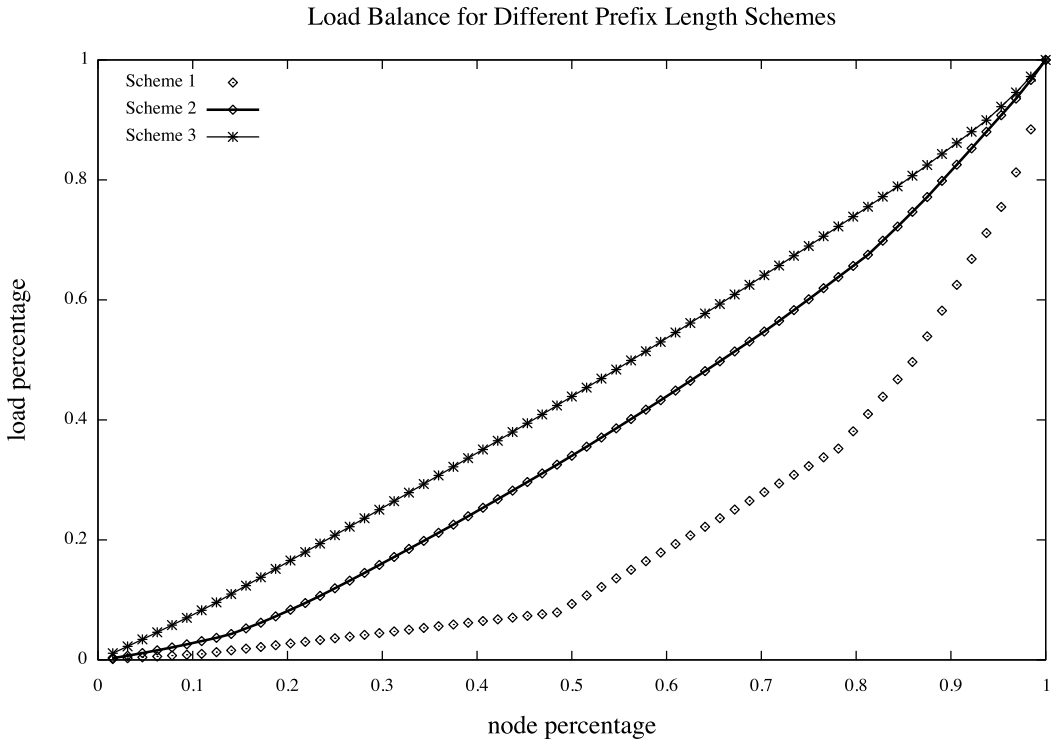


Figure 6.7: Load Balancing with Different Schemes

ber of objects handled by a given set of nodes divided by the total number of objects) for a given node percentage (i.e., the number of nodes in the set divided by the total number of nodes). A well balanced scheme should yield a linear relationship between the load percentage and the node percentage (where $y = x$, i.e., diagonal), meaning that each node receives the same number of objects to index. The farther the curve is away from the diagonal, the worse it is.

As we can see from Figure 6.7, when Scheme 1 was chosen as the length of prefix l, the load is not well balanced. The curve is far away from the diagonal and shows some saltations. Scheme 3 performs best among the three because it is very close to the diagonal, which implies that the load is well balanced. However, Scheme 3 makes l too long and the number of groups becomes too big, leading to

less objects in each group. This significantly affects the indexing cost. Overall, from our study, Scheme 2 provides a good choice for l, with which the work load is also well balanced.

There is a tradeoff between indexing performance and load balancing. To improve the former, l should be smaller, which leads to poor load balancing. To improve the latter, l should be bigger to ensure that all nodes have groups to be responsible for, which leads to a higher indexing cost. Scheme 2 shows acceptable results on both indexing performance and load balancing. In reality, we can choose different schemes for different scenarios. For example, if the performance of an application is very important, Scheme 1 will be a good choice.

6.4.2 Performance Study on the TISH Model

In this section, we present the experimental results for the TISH model that we proposed in Chapter 4. Since TISH is a model to capture the dynamicity of object moving patterns, its accuracy is important to the applications those use it as the foundation of further mining or assistance. So, the first experiment that we ran is to verify its accuracy for various networks within which the objects moving in different patterns. Also, in large-scale applications, the volume of data is huge. We want to see whether TISH can scale well and does not cost enormous bandwidth.

Experiments were conducted on a Core 2 Quad 2.40GHz machine with 4GB RAM. We simulated a network with 1,000 nodes. This network is built with the following characteristics: i) the network overlay is a connected unidirectional graph; ii) there are no hot or cold spots in the network; and iii) the fanout of the nodes follows normal distribution with a given average (see Figure 6.8) and variance (0.01). The first characteristic guarantees that every node is involved in

the experiments, so there are no outliers. The last two help to keep the variance of calculating averages low. The edges in the graph represents a business connection, along which objects move.

All nodes in the network have **V** objects of their own at the beginning of the experiments. Each connection is associated with a time-varying pattern from a pre-defined pattern set (see Table 6.9)[5]. All the nodes send objects to neighbors with the amount determined by the associated pattern. These patterns vary in the volume of object flow for time t (i.e., $g(t)$) where **V** is a constant coefficient and $random()$ returns a number which is within $(0, 1]$. If there are less objects than $g(t)$, the node generates enough objects to make up for the objects. Parameters a and b in the patterns are random numbers between 0 and 1 (inclusive), and are chosen before the experiments and remain constant during the experiments.

For each epoch, a node randomly chooses half of its connections to send objects. The connections with/without objects moving on inside an epoch are called *active connections*/*idle connections*, respectively.

An RFID object generator has been implemented to generate RFID objects for each pattern.The settings of the experiments are summarized in Table 6.8.

Parameter	Default Value
Number of Nodes	1000
Fanout	10
Number of Slots	10
Value of V, c and w_s	1000, 2, 10
Size of Sample (m)	50

Figure 6.8: Default Settings of Experiments for TISH

[5]It should be noted that "Random" is actually not a pattern, we use it to see how randomness affects the performance and accuracy of our model.

Pattern Name	Definition ($g(t)$)
Constant	$a * \mathbf{V}$ (a is chosen randomly)
Random	$\mathbf{V} * random()$
Segmentary	$a * \mathbf{V}$, if $2 * seg \leq t < 2 * seg + 1$ $b * \mathbf{V}$, if $2 * seg + 1 \leq t < (2 + 1) * seg$ seg is 100 Event Cycles a, b are chosen randomly
Sinusoidal	$\lvert\mathbf{V} * \sin(t)\rvert$

Figure 6.9: Patterns in Experiments for TISH

6.4.2.1 Accuracy of TISH

Since the TISH model keeps more information on recent data, we expect that the accuracy decreases for the distant data. However, the accuracy loss should not be significant. The error for the TISH model in a given time frame is defined as the difference between the real and the modeled distribution of objects' source nodes. To accurately represent this, we first calculated the average of δ (described in Section 4.5.1) as $\bar{\delta}$ for all neighbors at each node. We then calculated the error of the model as the average of $\bar{\delta}$ for all nodes in the system.

We ran the simulation using the default settings in Figure 6.8 for 1000 event cycles. During each cycle (called an *epoch*), several objects were sent from one node to another according to the pattern associated with this connection, if it is active. The objects are randomly chosen from local objects, including the ones initially assigned to a node and those sent to it by its neighbors. The experiments were done for each pattern separately (i.e., all connections are associated with the same pattern) and all patterns together (i.e., each connection is associated with a randomly chosen pattern). In this experiment, after the 1000 event cycles finished, we calculated the error ϵ, for each epoch.

The result is shown in Figure 6.10 and Figure 6.11. The time axis represents

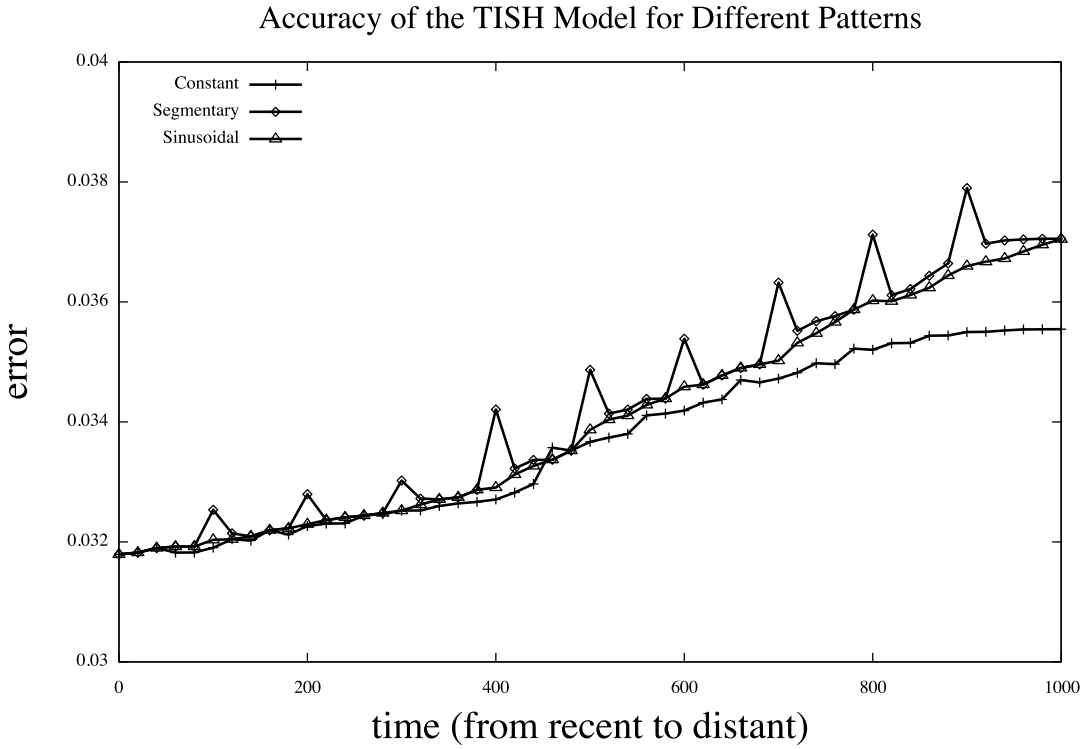


Figure 6.10: Accuracy of the Model for Different Patterns

the epochs from the most recent ($1st$) to the most distant ($1000th$).

Figure 6.10 shows the error of the TISH model in experiments that only a single pattern is chosen for all nodes in the network ("Random Pattern" is not included, as "Random" is actually not a pattern at all). We note that the error is very low. Although it increases for distant epochs, the increasing rate is not high. In theory, "Constant Pattern" should not generate any error because the distribution of objects are never changed. In practice, the sampling process and idle epochs add randomness to the system. For "Sinusoidal Pattern", although the volume changes, the distribution does not. So the orders of $\mathcal{B}$ and $\mathcal{B}'$ are not changed. This explains why it shows almost exactly same results with the "Constant Pattern". "Segmentary Pattern" shows an periodic pattern where

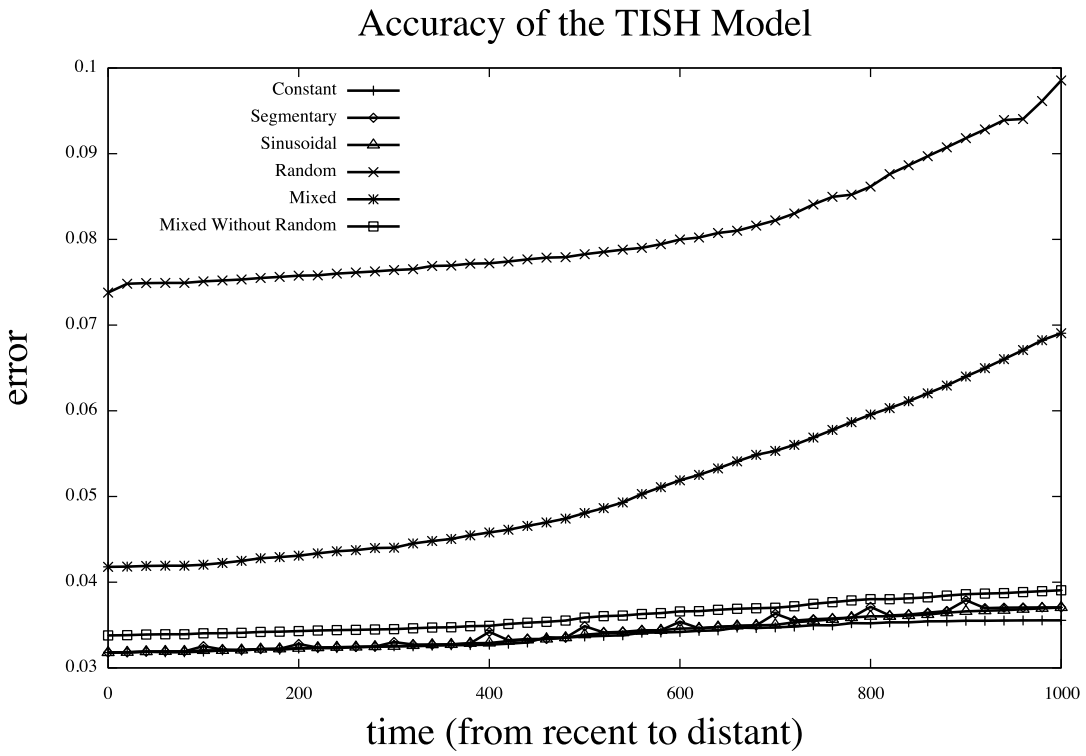


Figure 6.11: Accuracy of the Model with Mixed and Random Pattern

every 100 epochs, the error increases. This is caused by the change of patterns described in Figure 6.9. However, after the change finished, the error quickly decreases to almost the same with "Constant Pattern" again.

The impact of randomness on the accuracy of the TISH model is important. We ran the simulation with "Random Pattern" (all connections are associated with "Random Pattern"), reusing the settings in the experiments without "Random Pattern". We also ran the simulations for the scenarios of "Mixed" (each node randomly picked a pattern from Figure 6.9 individually) and "Mixed Without Random" (each node randomly picked a pattern from Figure 6.9 without "Random Pattern"). For "Mixed" or "Mixed Without Random", connections are associated with different patterns. Figure 6.11 adds results for "Random

Pattern", "Mixed Without Random" and "Mixed". The "Mixed Without Random" experiment shows that even when the nodes in the network choose different patterns for different connections, the TISH model is still able to describe them accurately. Compared to the "Constant Pattern", the extra error for the "Mixed Without Random" experiment is caused by the mixing "Segmentary Pattern" and "Sinusoidal Pattern". Since the change in "Sinusoidal Pattern" is continuous, at some point (when $\sin(t) = a$ or $\sin(t) = b$), the order of $\mathcal{B}$ is changed.

"Random Pattern" generates a high error because it is not a pattern and is unpredictable. Mixing it with other patterns also increases the average error.

6.4.2.2 The Cost of Model Maintenance

The main performance bottleneck in our model is caused by the procedure of querying neighbors for new incoming objects. It is also possible that when the new patterns are being established, more network calls are used due to the use of the underlying P2P overlay. However, as discussed in Section 4.4, our model is sensitive to these kind of network changes and adapts quickly with the changes.

In this experiment, we verified this rapid dynamic adaptation of the model by counting the number of network calls at different time points. The system setting is the same as the "Mixed without Random" one. Figure 6.12 shows the result. We can see that during the time of system bootstrap, the network traffic is higher. This is because at that time there was no history information and all the objects were found by P2P calls. However, after the TISH model has been established, only a few network calls are used and the number of network calls stays stable.

Another interesting performance evaluation is to investigate how many network calls are used per query for the model maintenance, and the type of distri-

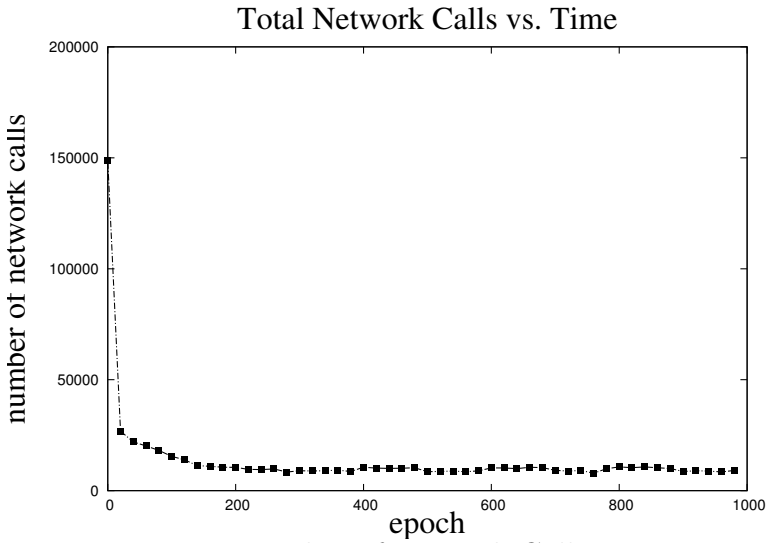


Figure 6.12: Number of Network Calls vs. Time

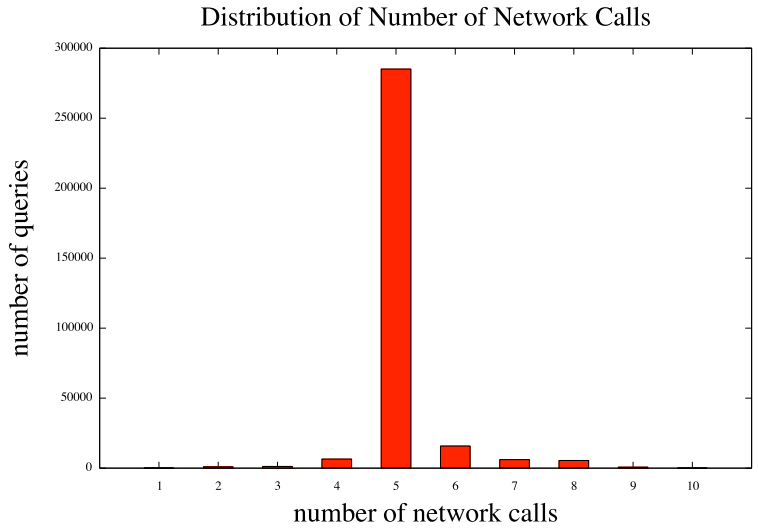


Figure 6.13: Distribution of Number of Network Calls for Model Maintenance

bution we obtain. We illustrate the result in Figure 6.13, by using a histogram. We note that the majority of the queries use less than 10 (the average number of fan-out) network calls in order to maintain the TISH model. According to the analysis in Section 4.5, we expect that the average number of network calls for model maintenance is close to the *effective fan-outs*, which is the average number

of active (non-idle) connections for all the nodes (5 in our experiments). The average number of network calls in this experiment is 5.11, which is close to what we expect.

6.4.3 Performance Study on the PeerTrack Cloud

The PeerTrack Cloud architecture aims at reducing the query processing time so that it can be used in time-constrained environments. In this section, we verify that the query time is significantly shorter than that in either the EPCglobal Architecture Framework or the P2P architecture. We conducted the experiments on the CloudSim simulation platform [CRB10]. In this section, we present the results of these experiments.

The queries that we used in these experiments are the same as the ones used in Section 6.4.1, Table 6.2. We simulated a network with 512 locations (nodes) and each node generated a certain number of objects at the beginning of the tests. The number of generated objects increases from 500 to 5,000, at the step of 500. Then we ran the system for 10,000 event cycles to build the object moving datasets[6]. Finally, we ran the queries and recorded the average response time.

The settings for the P2P and centralized solution are almost the same as the ones in Section 6.4.1[7], except that in this experiments, we randomly assigned a coordination to each location and assumed that the time of remote access between two locations is proportional to the distance between them. For the PeerTrack Cloud simulations, since the number of copies has serious influence on the performance, we ran the experiments with different configurations on this parameter. The number of data centers used in the simulation was 16, and they

[6]This process is the same as the one in Section 6.4.1

[7]The Discovery Service is implemented by a single database.

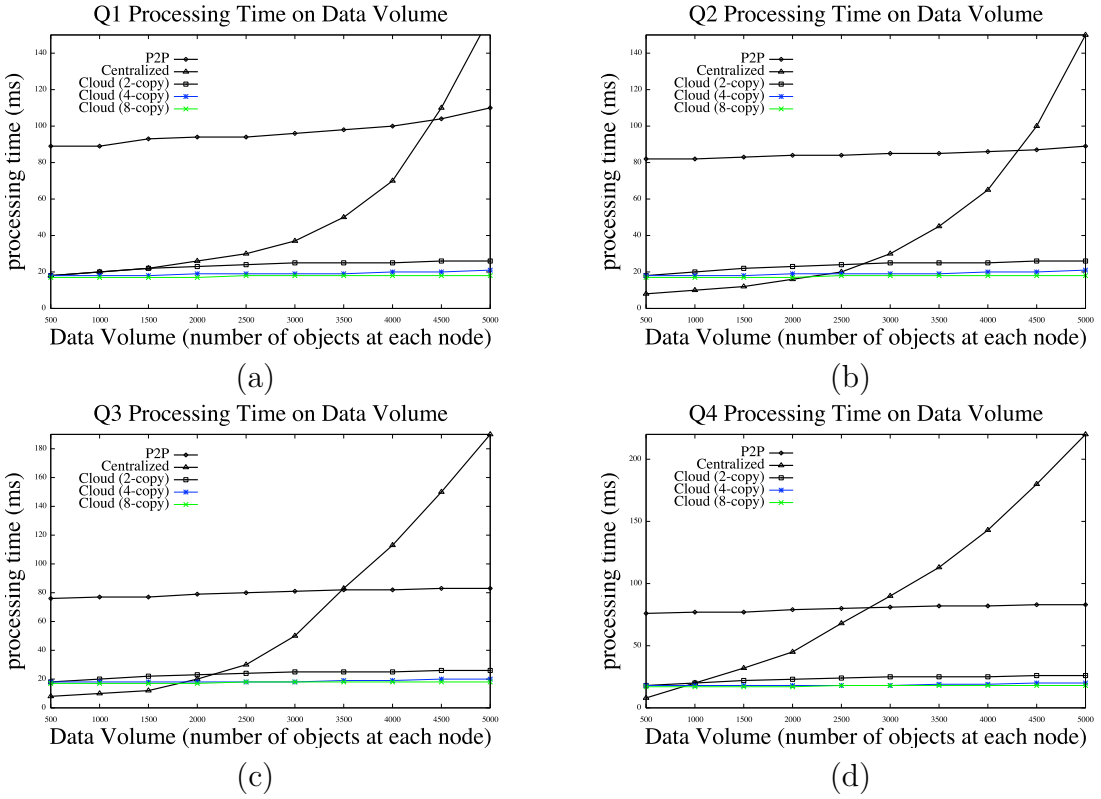


Figure 6.14: Query Processing Performance of PeerTrack Cloud

were distributed evenly around the globe.

The results of the experiments are shown in Figure 6.14. It illustrates two important characteristics of PeerTrack Cloud compared with the other two architectures.

Firstly, the response time for PeerTrack Cloud is barely influenced by the volume of data. This is explained by our analysis in Section 5.4. Since the object's location information is stored in the PT-LBS2, where the data is replicated at several data centers near those locations where the data is related to, the required number of remote access is mostly two. On the contrary, for the track and movement-based statistical queries (Q2 and Q3, respectively) in P2P architecture, we have to redirect the query to the related node, and for trace and path-based

statistical queries (Q1 and Q4, respectively), the response time is proportional to the length of the moving path. Thus, averagely, it stays the same. For the centralized solution, the response time is affected by how the database responds to the queries. The track queries can be answered by a select operation, which in most cases is fast. However, the other three queries all require join operation which needs much more time to execute. As a result, we can see from the figure that for non-track queries, the response time in centralized solution rises quickly.

Secondly, the number of copies affects the performance of PeerTrack Cloud. It is clear that the more copies, the quicker it answers traceability queries. From Section 5.4, we know that the response time in PeerTrack Cloud is $t_{PT-T2S} + \bar{t}_{DC}$ or $t_{PT-T2S} + min(t_{DC})$, respectively for tracking/tracing and statistical queries. The more copies, the closer a request is to a data center with the data, averagely. We can see that when there are 8 copies, the response time is very short and it stays almost the same, while when there are only 2 copies, it is longer. However, we cannot make unlimited number of copies, because of the hardware and bandwidth costs. This experiment suggests that if the application requires real-time access to the traceability data, more copies should be made. Otherwise, it is not necessary.

6.5 Summary

In this chapter, we have presented the implementation of our proposed PeerTrack Platform. To validate the feasibility and benefits of our proposed approaches, we developed an asset management system on the top of the platform and conducted an extensive set of performance studies. The findings can be summarized as follows. Firstly, the developed application illustrates that our system can provide not only interfaces for query processing, but also interfaces for rule-based event

processing in traceable RFID networks. Secondly, the extensive experiments reveal that the distributed architecture and MOODS model is more scalable and outperforms the centralized approach when the size of network or the volume of data is large. Thirdly, the TISH model can accurately represent the object moving patterns and quickly adapt to the changes of these patterns. Finally, the PeerTrack Cloud system is able to provide quick response in a time-critical environment.

Chapter 7

Conclusions

In this chapter, we summarize the contributions of this dissertation and discuss future research directions for enabling traceability in large-scale RFID networks.

7.1 Summary

In the recent years, RFID is emerging as a promising technology to enable automatic tracking and tracing in large, global business networks [CKR04a, SLZ08b, WRS11]. However, with growth in the deployment of RFID technology, new problems and challenges have been posed to application developers and service providers to support traceability with RFID in large-scale traceable networks. Adequate solutions to the new problems will be very important for realizing traceability in a scalable, flexible and affordable way.

In this dissertation, we have proposed a generic P2P architecture and distributed data models for large traceable RFID networks. We also proposed a cloud-based architecture which is built on the top of the distributed data models to provide real-time response to traceability queries. We implemented our approaches in the PeerTrack platform and developed a real-world applications based on it. In particular, we summarize our main research contributions in the following:

- **Scalable, Privacy-Aware and Efficient P2P Architecture and Traceability Model:** We proposed a generic P2P traceable network architecture where the partners in the system do not have to share their data in a centralized database, or in any other open ways. Instead, the partners can choose which portion of their data to share, and to whom. This is very important for business with confidential information. On the other hand, in order to enable item-level traceability, the data much be shared in some way. Our architecture features the fully-anonymous indexing that distributes the index into the whole network, so that nobody can access all the data from any partner. This architecture is built on the top of a distributed traceability model, namely MOODS. MOODS is essentially a distributed linked list, where the nodes in the same moving path is connected by the source and destination attributes. We proposed the algorithms to maintain these information so that this model can be used to answer item-level tracking and tracing queries, as well as statistical queries. These algorithms are carefully designed so that the workload among all the nodes are well balanced, meanwhile the bandwidth usage is optimized.

- **Mining Model for Object Moving Patterns:** RFID streams have implicit information about the patterns of objects' movements. Discovering these patterns and capturing their changes quickly can help the business managers to make more rational decisions. We proposed a distributed mining model, namely TISH, which is a combination of two important mining tools, namely Titled Time Frame and Histogram. This model is able to maintain the history of moving patterns between nodes for a long time with small memory footprint. We also proposed the distributed algorithms to maintain the titled time frame of histograms on the top of the P2P ar-

chitecture. The input is firstly sampled to save the bandwidth cost for the maintenance. And we make use of the recent information about the patterns to guide the rewriting of queries to further reduce bandwidth costs. This model can be used in various mining tasks.

- **Cloud-based Infrastructure to Enable Real-time Query Response:** Some applications requires the query to be processed in a timely manner. In P2P architectures, this is hard to achieve due to the fact that P2P access often requires several intermediate hops in the network. Cloud computing systems replicate the data to several locations so that it can lead a request to the nearest data center that can answer it. It provides much faster response that other architectures such as P2P or centralized solutions. However, existing cloud computing architectures are too generic to take the advantage of the characteristics of RFID data, which is location-based. We proposed a new cloud infrastructure which is designed specifically for traceable RFID network. This infrastructure features several loose-coupled modules which can be provided to the users separately. Its location-based storage service is designed to partition and replicate the traceability data according to their location information. In this way, information about a location is stored close to it, so that the queries can be answered efficiently.

- **Implementation and Performance Study:** We implemented these techniques inside the PeerTrack Platform. We adopted a number of state-of-the-art technologies for the implementation. Our implementation has led to a comprehensive platform for enabling traceability in large-scale networks. We proved this by a real-world application that has been developed and deployed at a local company. We studied the performance of the proposed techniques with extensive experiments, including those for scalability,

workload balancing, bandwidth usage and response time.

7.2 Future Directions

There are still some research issues need to be addressed, according to our overview of the state-of-the-art development of traceability applications in Chapter 2. This dissertation focuses on providing a privacy-aware, scalable and efficient system, meanwhile the other important issues are left for future work. In particular, we identify the following directions for future research: uncertainty management, realizing the Internet of Things and complex event processing.

- **Uncertainty Management:** While RFID provides promising benefits in many applications, there remains significant challenges to be overcome before these benefits are realized. Central to these challenges is the uncertainty of the data collected by the underlying RFID networks. Although current RFID reader accuracy is improving, erroneous readings still occur in RFID systems such as duplicated reads, missed and incorrect reads due to interference or malfunction of RFID components. Raw RFID data are therefore typically incomplete, imprecise, and even misleading [JFG08]. When such data streams are used directly in monitoring and tracking applications (e.g., product recall), the quality of the applications is often a significant concern. Development of techniques for effectively managing uncertainty in RFID traceability networks gains further significance in that it offers the opportunity to ensure high-quality of many other innovative and high payoff applications. In particular, we want to design a system that can transform the uncertain information to probabilistic data with a certain confidential margin. Also, a database management system should be

designed to store this kind of information and provide fast access methods for them.

- **Realizing Internet of Things:** Internet of Things (IoT) is a global network where everyday objects such as buildings, sidewalks, and commodities are identifiable, readable, addressable, and even controllable via the Internet [KS10]. Such a ubiquitous network offers the capability of integrating the information from both the physical world and the virtual one, which not only affects the way how we live, but also creates tremendous business opportunities such as efficient supply chains, independent living of elderly persons, and improved environmental monitoring. Traceability is one of IoT's key features. However, there are still some other challenges. For example, RFID can only be used to trace objects' locations at certain observation points. In order to realize full-time ubiquitous monitoring, other technologies such as GPS, sensors must be seamlessly integrated into the system.

- **Complex Event Processing:** As we discussed in Chapter 2, there are some events that we cannot process at this moment. In particular, without the help of specially designed hardware, it is impossible to capture the containment relationship and the change of that relationship. For example, at a distribution center, goods from different containers may be re-packed into one container. Regular hardware can only capture the goods themselves, but not the change of containment relationship. This brings some troubles in business management. It is impossible to tell whether a product is going to be delivered to a wrong place until it arrives. Capturing these kinds of complex events and processing them efficiently is a difficult task and extensive research is needed [FJK05, JAF06].

Appendix A

Curriculum Vitae

Personal Information

Name:	Yanbo Wu
Tel:	+61 8 8313 4728
Email:	yanbo.wu@adelaide.edu.au
Home Page:	http://www.cs.adelaide.edu.au/∼yanbo

Research Interests

RFID and Sensor Networks, Internet of Things, Distributed and Parallel Databases, Data Mining, World Wide Web, Social Networks

Highlights

1. Published refereed technical papers in: i) top journals including Distributed and Parallel Databases, Computer Journal, Data and Knowledge Engineering; and ii) international conferences such as ICPP'11, AINA'10.

2. Participated actively in academic activities: i) organized IWRT'11, acted as program committee for APWeb'12, and ii) served as an external reviewer

for many journals and conferences.

3. Interned at IBM Almaden Research Center, 2010-2011. Worked on text search and helped to improve the performance by 90%.

4. Awarded Google Top-up Grant, 2011. IBM Travel Award for ICSOC, 2008

5. Won the championship of TopCoder Open 2009.

6. Advanced programming experience in Java, Ruby and Object-C.

Awards, Honors, Fellowships

Oct 2011	Google Top-up Grant
May 2009	TopCoder Open 2009 Component Development Champion
Mar 2009	TopCoder "Coder of the Month"
Jun 2006	AFSI Scholarship of The University of Adelaide
Jun 2006	ARC Grant-funded Scholarship
2004	Tsinghua Friends - Zhifu Chen Scholarship
2003	Tsinghua Friends - Geru Zheng Scholarship

Education

2008-2011 Ph.D. in Computer Science
School of Computer Science
The University of Adelaide. Adelaide, Australia
Dissertation: *Enabling Traceability in Large-Scale RFID Networks*
Supervisor: Dr. Quan Z. Sheng

2001-2005 B.E. in Computer Science
Tsinghua University (THU). Beijing, China

1998-2001 High School Diploma
No. 6 High School. Laizhou, China
Final rank as top 1
Top 3 in the entrance test for college in Shandong Province

Research Experience

Jun 2008-present	PhD Student at The University of Adelaide Work on enabling traceability in traceable RFID networks. I designed and implemented the PeerTrack Platform. This work was published in several conference papers including ICPP'11, AINA'10, WAMIS'11, together with journal papers including Distributed and Parallel Databases, Data and Knowledge Engineering and Computer Journal.
Aug 2010-Feb 2011	Research intern at IBM Almaden Research Center (ARC) Worked on text search performance.
Jun 2005-Oct 2005	Research associate at City University of Hong Kong Participated in the research of Anti-Phishing. Developed the commercial software "Reasonable Antiphishing" (`http://reasonables.com/antiphishing.aspx`).
May 2003-Oct 2003	Research intern at IBM China Research Lab (CRL) Work on natural language processing algorithms.

Industrial Experience

Nov 2006-Feb 2008	Component Manager at TopCoder Inc. I managed the competitions at TopCoder Inc and maintained the Online Review system at http://www.topcoder.com/tc.
Nov 2005-Oct 2006	Freelancer at TopCoder Inc. Competed for TopCoder projects and won 15 projects with average score of 93 and 55.56% winning rate. Projects included J2EE systems for Fortune 500 companies and TopCoder UML Tool (Java Swing).
May 2005-Feb 2006	Software Developer at Roxbeam Inc. Developed a P2P Video-on-Demand system.

Publications

Refereed journal and conference publications

1. "Facilitating Efficient Object Tracking in Large Scale Networks". Yanbo Wu, Quan Z. Sheng and Damith Ranasinghe. Computer Journal, 2011 (to appear)

2. "A Semantically Enhanced Service Repository for User-Centric Service Discovery and Management". Jian Yu, Quan Z. Sheng, Jun Han, and Yanbo Wu. Data and Knowledge Engineering, 2011 (to appear)

3. "RFID Enabled Traceability Networks: A Survey". Yanbo Wu, Damith Ranasinghe, Quan Z. Sheng, Sherali Zeadally and Jian Yu. Distributed and Parallel Databases. Vol 29, No 5-6, pp 397-443. 2011

4. "Peer-to-Peer Objects Tracking in the Internet of Things". Yanbo Wu, Quan Z. Sheng and Damith Ranasinghe. Proceedings of the International Conference on Parallel Processing (ICPP'11). Taipei, Taiwan. September 12-17, 2011

5. "Tracing Moving Objects in Internet-based RFID Networks". Yanbo Wu, Quan Z. Sheng and Damith Ranasinghe. Proceedings of the International Symposium on Web and Mobile Information Systems (WAMIS'11). Singapore, Singapore. March 22-25, 2011

6. "Enabling Scalable RFID Traceability Networks". Quan Z. Sheng, Yanbo Wu and Damith Ranasinghe. Proceedings of the International Conference on Advanced Information Networking and Applications (AINA'10). Perth, Australia.

April 20-23, 2010

7. "Realizing the Internet of Things in Service-Centric Environments". Yanbo Wu. Proceedings of the International Conference on Service Oriented Computing (ICSOC'08). Sydney, Australia. December, 2008

Papers submitted/in preparation

8. "Modeling Object Flows from Distributed and Sovereign RFID Data Streams for Efficient Tracking and Tracing". Yanbo Wu, Quan Z. Sheng, Hong Shen and Sherali Zeadally. IEEE Transactions on Parallel and Distributed Systems. (submitted)

9. "Discovering Object Moving Patterns in the Internet of Things". Yanbo Wu and Quan Z. Sheng. International Conference on Advanced Information Systems Engineering (CAiSE'12). (submitted)

10. "Modeling Sovereign RFID Data Streams in Collaborative Traceable Networks". Yanbo Wu and Quan Z. Sheng. International Conference on Extending Database Technology (EDBT'12). (submitted)

11. "PeerTrack: A Platform for Tracking and Tracing Objects in Large-Scale Traceability Networks". Yanbo Wu, Quan Z. Sheng and Damith Ranasinghe. International Conference on Extending Database Technology (EDBT'12). (submitted)

Formal Presentations

1. Conference talk at ICPP'11, "Peer-to-Peer Objects Tracking in the Internet of Things", 14 September 2011, Taipei, Taiwan

2. Conference talk at AINA'10, "Enabling Scalable RFID Traceability Networks", 20 April 2010, Perth, Australia

3. Conference talk at ICSOC'08, "Realizing the Internet of Things in Service-Centric Environments", 1 December 2008, Sydney, Australia

Professional Services

1. Program Committee Member, The 14th Asia-Pacific Web Conference (APWeb'12), Kunming, China, April 11-13, 2012

2. Program Chair, International Workshop on RFID Technology - Concepts, Applications, and Challenges (IWRT'11), Niagara Falls, Ontario, Canada, September 19-21, 2011

3. Publicity Co-Chair, International Workshop on RFID Technology - Concepts, Applications, and Challenges (IWRT'09), Milan, Italy, 6-7 May, 2009

4. External reviewer for journals like Information System Frontiers, Computer Communications Journal, International Journal of Computational Sciences, IEEE Communications Letters.

Teaching Experience

2009	Tutor for *C++ Programming for Engineers*
2008-2009	Tutor for *Introduction to Software Engineering*
2005	Tutor for *Java Programming Language*
2009	Honours thesis supervision Xin Tang (Bachelor of Software Engineering) Thesis: *Implementation of a Peer-based RFID Event Processing Framework*
2009	Honours thesis supervision Yvonne Teo (Bachelor of Software Engineering) Thesis: *Intelligent Returnable Asset Management System*
2008	Summer school project supervision Xin Tang Thesis: *TnT : A Query Language for Tracing and Tracking Movable Objects*
2009	Summer school project supervision Yuguo Li (Bachelor of Software Engineering) Thesis: *PeerTrack Platform Evaluation*

Skills

1. Proficient in RFID, Database Management Systems (DBMS).

2. Familiar with UML and model driven approach.

3. Languages: proficient in JAVA, Ruby, Object-C, C#; familiar with C/C++, HTML, Javascript.

4. Databases: MySQL, Informix, Oracle, Drizzle.

5. Operating systems: Linux, Windows, Mac OSX.

Bibliography

[ABS06] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha Nabar, Tomoe Sugihara, and Jennifer Widom. "Trio: a System for Data, Uncertainty, and Lineage." In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB'06)*, Seoul, Korea, 2006.

[ACK06a] Rakesh Agrawal, Alvin Cheung, Karin Kailing, and Stefan Schonauer. "Towards Traceability across Sovereign, Distributed RFID Databases." In *Proceedings of the 10th International Database Engineering and Applications Symposium (IDEAS'06)*, Delhi, India, 2006.

[ACK06b] Rakesh Agrawal, Alvin Cheung, Karin Kailing, and Stefan Schönauer. "Towards Traceability Across Sovereign, Distributed RFID Databases." In *Proceedings of the 10th Intl. Database Engineering and Applications Symposium (IDEAS'06)*, Delhi, India, December 2006.

[AF05] Manfred Aigner and Martin Feldhofer. "Secure Symmetric Authentication for RFID Tags." In *Proceedings of the Telecommunication and Mobile Computing (TCMC'05)*, Graz, Austria, 2005.

[AFG09] Michael Armbrust, Armando Fox, Rean Griffit, Anthony D. Joseph, Randy H. Kat, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. "Above the Clouds: A Berkeley View of Cloud Computing." Technical report, Electrical Engineering and Computer Sciences, University of California at Berkeley, February 2009.

[AFG10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. "A view of cloud computing." *Communications of ACM*, **53**:50–58, April 2010.

[AGG] Keith Alexander, Tig Gilliam, Kathy Gramling, and Chris Grubelic. "Applying Auto-ID to Reduce Losses Associated with Shrink." `http://www.autoidlabs.org/uploads/media/IBM-AUTOID-BC-003.pdf`.

[AGP00] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. "Congressional Samples for Approximate Answering of Group-by Queries."

In *Proceedings of the 2000 ACM International Conference on Management Of Data (SIGMOD'00)*, Dallas, Texas, USA, 2000.

[AH11] Leonardo Albernaz Amaral and Fabiano Hessel. "Cooperative CEP-based RFID Framework: A Notification Approach for Sharing Complex Business Events Among Organizations." In *Proceedings of the 5th IEEE International Conference on RFID (RFID'11)*, Orlando, Florida, April 2011.

[AKS02] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. "Hippocratic Databases." In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*, 2002.

[Ang05] Rebecca Angeles. "RFID Technologies: Supply-Chain Applications and Implementation Issues." *Information Systems Management*, **22**(1):51–65, December 2005.

[BBD02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. "Models and Issues in Data Stream Systems." In *Proceedings of the 21th ACM Symposium on Principles of Database Systems (PODS'02)*, Madison, Wisconsin, 2002.

[BCD03] Brian Babcock, Surajit Chaudhuri, and Gautam Das. "Dynamic Sample Selection for Approximate Query Processing." In *Proceedings of the 2003 ACM International Conference on Management Of Data (SIGMOD'03)*, San Diego, California, 2003.

[BCD11] Philip A. Bernstein, Istvan Cseri, Nishant Dani, Nigel Ellis, Ajay Kalhan, Gopal Kakivaya, David B. Lomet, Ramesh Manne, Lev Novik, and Tomas Talius. "Adapting Microsoft SQL Server for Cloud Computing." In *Proceedings of the 27th International Conference on Data Engineering (ICDE 2011)*, Hannover, Germany, April 2011.

[BHK07] I. Baumgart, B. Heep, and S. Krause. "OverSim: A Flexible Overlay Network Simulation Framework." In *Proceedings of the 10th Global Internet Symposium (GI 2007)*, Anchorage, USA, 2007.

[BJ01] John D. Kubiatowicz Ben Y. Zhao and Anthony D. Joseph. "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing." Technical report, Berkeley, CA, USA, 2001.

[BKK03] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. "Looking up Data in P2P Systems." *Communications of the ACM*, **46**(2):43–48, 2003.

[BMN08] V. Botea, D. Mallett, M. A. Nascimento, and J. Sander. "PIST: An Efficient and Practical Indexing Technique for Historical Spatio-Temporal Point Data." *Geoinformatica*, **12**(2):143–168, 2008.

[BO05] L. M. Buckley and C. W. Olson. "High Tech, High Stakes: Using Technology to Smash the Fake Trade." *IPWorld*, pp. 30–33, May 2005.

[CAA01] H. Chon, D. Agrawal, and A. Abbadi. "Storage and Retrieval of Moving Objects." In *Proceedings of MDM'01*, pp. 173–184, Hong Kong, China, 2001. Springer-Verlag, London, UK.

[CCG11] Yu Cao, Chun Chen, Fei Guo, Dawei Jiang, Yuting Lin, Beng Chin Ooi, Hoang Tam Vo, Sai Wu, and Quanqing Xu. "ES2: A Cloud Data Storage System for Supporting Both OLTP and OLAP." In *Proceedings of the 27th International Conference on Data Engineering (ICDE 2011)*, Hannover, Germany, April 2011.

[CE11] Craig R. Carter and P. Liane Easton. "Sustainable Supply Chain Management: Evolution and Future Directions." *International Journal of Physical Distribution and Logistics Management*, **41**(1):46–62, 1 2011.

[CGP10] José J. Cantero, Miguel A. Guijarro, Antonio Plaza, Guillermo Arrebola, and Janie Baños. "A Design for Secure Discovery Services in the EPCglobal Architecture." In *Unique Radio Innovation for the 21st Century: Building Scalable and Global RFID Networks*. Springer, 2010.

[Cho11] Tsan-Ming Choi. "Coordination and Risk Analysis of VMI Supply Chains With RFID Technology." *Industrial Informatics, IEEE Transactions on*, **7**(3):497–504, 8 2011.

[CJLar] Hanhua Chen, Hai Jin, Xucheng Luo, Yunhao Liu, Tao Gu, Kaiji Chen, and Lionel Ni. "BloomCast: Efficient and Effective Full-Text Retrieval in Unstructured P2P Networks." *IEEE Transactions on Parallel and Distributed Systems*, 2011 (to appear).

[CKR04a] Sudarshan S. Chawathe, Venkat Krishnamurthy, Sridhar Ramachandran, and Sanjay Sarma. "Managing RFID Data." In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, Toronto, Canada, September 2004.

[CKR04b] Sudarshan S. Chawathe, Venkat Krishnamurthy, Sridhar Ramachandran, and Sanjay Sarma. "Managing RFID Data." In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, Toronto, Canada, September 2004.

[CKS07a] Alvin Cheung, Karin Kailing, and Stefan Schönauer. "Theseos: A Query Engine for Traceability Across Sovereign, Distributed RFID Databases." In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*, Istanbul, Turkey, 2007.

[CKS07b] Alvin Cheung, Karin Kailing, and Stefan Schönauer. "Theseos: A Query Engine for Traceability Across Sovereign, Distributed RFID Databases." In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*, Istanbul, Turkey, 2007.

[Coc07] Richard Cocci. "SPIRE: Scalable Processing of RFID Event Streams." In *Proceedings of the 5th RFID Academic Convocation*, Brussels, Belgium, 2007.

[CRB10] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov1, Csar A. F. De Rose, and Rajkumar Buyya. "OverSim: A Flexible Overlay Network Simulation Framework." *Software: Practice and Experience*, **41**(1):2350, 2010.

[CSD11] Zhao Cao, Charles Sutton, Yanlei Diao, and Prashant Shenoy. "Distributed Inference and Query Processing for RFID Tracking and Monitoring." *VLDB Endowment*, **4**, February 2011.

[CSP05] Reynold Cheng, Sarvjeet Singh, and Sunil Prabhakar. "U-DBMS: a Database System for Managing Constantly-evolving Data." In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, 2005.

[CTD08] Richard Cocci, Thanh Tran, Yanlei Diao, and Prashant Shenoy. "Efficient Data Interpretation and Compression over RFID Streams." In *Proceedings of the 24th International Conference on Data Engineering (ICDE'08)*, Cancun, Mexico, 2008.

[DIA] DIALOG. "Distributed Information Architectures for cOllaborative loGistics." http://dialog.hut.fi/.

[Dim05] Tassos Dimitriou. "A Lightweight RFID Protocol to Protect against Traceability and Cloning Attacks." In *Proceedings of the 1st International Conference on Security and Privacy for Emerging Areas*

in Communications Networks (SECURECOMM'05), Athens, Greece, 2005.

[DMS07] Thomas Diekmann, Adam Melski, and Matthias Schumann. "Data-on-Network vs. Data-on-Tag: Managing Data in Complex RFID Environments." In *Annual Hawaii International Conference on System Sciences (HICSS 2007)*, Honolulu, Hawaii, Jan. 2007.

[DOL07] R. Derakhshan, M.E. Orlowska, and Xue Li. "RFID Data Management: Challenges and Opportunities." In *Proceedings of the 1st IEEE International Conference on RFID (RFID'07)*, Vienna, Austria, march 2007.

[DRT01] Nicole DeHoratius, Ananth Raman, and Zeynep Ton. "Execution: The Missing Link in Retail Operations." *California Management Review*, **43**(3):136–151, April 2001.

[EGS98] Martin Erwig, Ralf Hartmut Güting, Markus Schneider, and Michalis Vazirgiannis. "Abstract and Discrete Modeling of Spatio-Temporal Data Types." In *Proceedings of GIS'98*, pp. 131–136, Washington, D.C., United States, 1998. ACM, New York USA.

[EPC] EPCGLOBAL. "EPCGLOBAL." `http://www.EPCGLOBAL.com`.

[EV95] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[FDA] FDA. "Combating Counterfeit Drugs, A Report of the Food and Drug Administration." `http://www.fda.gov/oc/initiatives/counterfeit/report02_04.html`.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. "Strong Authentication for RFID Systems Using the AES Algorithm." In *Proceedings of the 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'04)*, Cambridge, USA, 2004.

[Fin03] Klaus Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley & Sons, Inc, 2003.

[FJK05] Michael J. Franklin, Shawn R. Jeffery, Sailesh Krishnamurthy, Frederick Reiss, Shariq Rizvi, Eugene Wu, Owen Cooper, Anil Edakkunni, and Wei Hong. "Design Considerations for High Fan-in Systems: The

HiFi Approach." In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, USA, January 2005.

[FN09] Kary Främling and Jan Nyman. "From Tracking with RFID to Intelligent Products." In *Proceedings of 14th IEEE International Conference on Emerging Technologies and Factory Automation*, Palma de Mallorca, Spain, 2009.

[GCL08] Y. Gao, G. Chen, Q. Li, B. Zheng, and C. Li. "Processing Mutual Nearest Neighbor Queries for Moving Object Trajectories." In *Proceedings of MDM'08*, pp. 2176–2195, Beijing, China, 2008. Elsevier Science Inc., New York, USA.

[GHC10] Hector Gonzalez, Jiawei Han, Hong Cheng, Xiaolei Li, Diego Klabjan, and Tianyi Wu. "Modeling Massive RFID Data Sets: A Gateway-Based Movement Graph Approach." *IEEE Transactions on Knowledge and Data Engineering*, **22**:90–104, 2010.

[GHL06a] Hector Gonzalez, Jiawei Han, Xiaolei Li, and Diego Klabjan. "Warehousing and Analyzing Massive RFID Data Sets." In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, USA, 2006.

[GHL06b] Hector Gonzalez, Jiawei Han, Xiaolei Li, and Diego Klabjan. "Warehousing and Analyzing Massive RFID Data Sets." In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, Georgia, USA, April 2006.

[GHP03] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, and Philip S. Yu. *Next Generation Data Mining*. AAAI/MIT, 2003.

[Gov] California Government. "California Business and Professions Code Sections 4163." `http://www.leginfo.ca.gov/calaw.html`.

[GS1] GS1. "GS1 Traceability." `http://www.gs1.org/productssolutions/traceability`.

[GWT09] Tao Gu, Zhanqing Wu, Xianping Tao, Hung Keng Pung, and Jian Lu. "epSICAR: An Emerging Patterns Based Approach to Sequential, Interleaved and Concurrent Activity Recognition." In *IEEE International Conference on Pervasive Computing and Communications*, Los Alamitos, CA, USA, 2009.

[GYG09] Yu Gu, Ge Yu, Na Guo, and Yueguo Chen. "Probabilistic Moving Range Query over RFID Spatio-temporal Data Streams." In *Proceeding of the 18th ACM conference on Information and Knowledge Management (CIKM'09)*, Hong Kong, China, 2009.

[HFS08] T. Huynh, M. Fritz, and B. Schiele. "Discovery of Activity Patterns Using Topic Models." In *Proceedings of the 10th International Conference on Ubiquitous Computing (Ubicomp '08)*, Seoul, South Korea, 2008.

[HHL03] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. "Querying the internet with PIER." In *Proceedings of VLDB'03*, pp. 321–332. VLDB Endowment, 2003.

[HK06] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Elsevier, 2006.

[HRZ08] K. Hose, A. Roth, A. Zeitz, K. Sattler, and F. Naumann. "A Research Agenda for Query Processing in Large-Scale Peer Data Management Systems." *Information Systems*, **33**(7-8):597–610, 2008.

[HSC05] Ying Hu, Seema Sundara, Timothy Chorma, and Jagannathan Srinivasan. "Supporting RFID-Based Item Tracking Applications in Oracle DBMS Using a Bitmap Datatype." In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, Trondheim, Norway, September 2005.

[IAM09a] Alexander Ilic, Thomas Andersen, and Florian Michahelles. "Increasing Supply-Chain Visibility with Rule-Based RFID Data Analysis." *IEEE Internet Computing*, **13**(1):31–38, January-February 2009.

[IAM09b] Alexander Ilic, Thomas Andersen, and Florian Michahelles. "Increasing Supply-Chain Visibility with Rule-Based RFID Data Analysis." *IEEE Internet Computing*, **13**(1):31–38, January-February 2009.

[JAF06] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom. "Declarative Support for Sensor Data Cleaning." In *Proceedings of the 4th International Conference on Pervasive Computing (Pervasive 2006)*, Dublin, Ireland, May 2006.

[JFG08] Shawn R. Jeffery, Michael J. Franklin, and Minos Garofalakis. "An Adaptive RFID Middleware for Supporting Metaphysical Data Independence." *The VLDB Journal*, **17**:265–289, March 2008.

[JLO04] C. S. Jensen, D. Lin, and B. Ooi. "Query and Update Efficient B+-tree Based Indexing of Moving Objects." In *Proceedings of VLDB'04*, pp. 768–779, Toronto, Canada, 2004. VLDB Endowment.

[JM04] M. Jelasity and A. Montresor. "Epidemic-Style Proactive Aggregation in Large Overlay Networks." In *Proceedings of ICDCS'04*, pp. 102–109, Tokyo, Japan, 2004. IEEE Computer Society, Washington DC, USA.

[JP02] Ari Juels and Ravikanth Pappu. "Squealing Euros: Privacy Protection in RFID-Enabled Banknotes." In *Financial Cryptography*, pp. 103–121. Springer-Verlag, 2002.

[Jue06] Ari Juels. "RFID Security and Privacy: a Research Survey." *IEEE Journal on Selected Area in Communications*, **24**(2):381–394, February 2006.

[KBM06] Thomas Kelepouris, Tom Baynham, and Duncan McFarlane. "Track and Trace Case Studies Report." `http://www.autoidlabs.org/uploads/media/AUTOIDLABS-WP-BIZAPP-035.pdf`, 2006.

[KCW09] Jinoh Kim, A. Chandra, and J.B. Weissman. "Using Data Accessibility for Resource Selection in Large-Scale Distributed Systems." *IEEE Transactions on Parallel and Distributed Systems*, **20**(6):788 – 801, june 2009.

[KF08] Michael Ketzenberg and Mark Ferguson. "Managing Slow Moving Perishables in the Grocery Industry." *Production and Operations Management*, **17**(5):513–521, September-October 2008.

[KMS10] Andreas Klein, Christian Mannweiler, Joerg Schneider, and Hans D. Schotten. "Access Schemes for Mobile Cloud Computing." *Proceedings of the 11th IEEE International Conference on Mobile Data Management (MDM 2010)*, May 2010.

[KPD07] Thomas Kelepouris, Katerina Pramatari, and Georgios Doukidis. "RFID-enabled traceability in the food supply chain." *Industrial Management and Data Systems*, **107**(2):183–200, 1 2007.

[KS10] Noboru Koshizuka and Ken Sakamura. "Ubiquitous ID: Standards for Ubiquitous Computing and the Internet of Things." *IEEE Pervasive Computing*, **9**(4):98 –101, october-december 2010.

[Lan05] Jeremy Landt. "The History of RFID." *IEEE Potentials*, **24**(4):8–11, October 2005.

[LC08] Chun-Hee Lee and Chin-Wan Chung. "Efficient Storage Scheme and Query Processing for Supply Chain Management Using RFID." In *Proceedings of the 28th ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, Vancouver, Canada, 2008.

[LC11] Chun-Hee Lee and Chin-Wan Chung. "RFID Data Processing in Supply Chain Management Using a Path Encoding Scheme." *IEEE Transactions on Knowledge and Data Engineering*, **23**(5):742–758, 2011.

[LDR08] R. Lange, F. Dürr, and K. Rothermel. "Scalable Processing of Trajectory-based Queries in Space-partitioned Moving Objects Databases." In *Proceedings of GIS'08*, pp. 31:1–31:10, Irvine, USA, 2008. ACM, New York, USA.

[LEB07] Dan Lin, Hicham G. Elmongui, Elisa Bertino, and Beng Chin Ooi. "Data Management in RFID Applications." In *Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA'07)*, Regensburg, Germany, 2007.

[LKG03] X. Li, Y. Kim, R. Govindan, and W. Hong. "Multi-dimensional Range Queries in Sensor Networks." In *Proceedings of SenSys'03*, pp. 63–75, Los Angeles, USA, 2003. ACM, New York, USA.

[LLSar] Xue Li, Jing Liu, Quan Z. Sheng, Sherali Zeadally, and Weicai Zhong. "TMS-RFID: Temporal Management of Large-scale RFID Applications." *Information Systems Frontiers*, 2011 (to appear).

[LP08] Dongmyung Lee and Jinwoo Park. "RFID-based Traceability in the Supply Chain." *Industrial Management and Data Systems*, **108**(6):713–725, 2008.

[LWL06] Shaorong Liu, Fusheng Wang, and Peiya Liu. "Integrated RFID Data Modeling: an Approach for Querying Physical Objects in Pervasive Computing." In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, Arlington, Virginia, USA, 2006.

[LXN07] Yunhao Liu, Li Xiao, and Lionel Ni. "Building a Scalable Bipartite P2P Overlay Network." *IEEE Transactions on Parallel and Distributed Systems*, **18**:1296–1306, 2007.

[LZ05] W. Lee and B. Zheng. "DSI: A Fully Distributed Spatial Index for Location-Based Wireless Broadcast Services." In *Proceedings of ICDCS'05*, pp. 349–358, USA, 2005. IEEE Computer Society, Los Alamitos, CA.

[LZZ09] Xuhui Li, Hao Zhang, and Yongfa Zhang. "Deploying Mobile Computation in Cloud Service." In *Proceedings of the 1st International Conference on Cloud Computing (CloudCom 2009)*, Beijing, China, November 2009.

[MGA03] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. "Spatio-Temporal Access Methods." *IEEE Data Engineering Bulletin*, **26**:40–49, 2003.

[MS05] A. Meka and A. Singh. "DIST: A Distributed Spatio-Temporal Index Structure for Sensor Networks." In *Proceedings of CIKM'05*, pp. 139–146, Bremen, Germany, 2005. ACM, New York, USA.

[MXA04] M. F. Mokbel, X. Xiong, and W. G. Aref. "SINA: Scalable Incremental Processing of Continuous Queries in Spatio-Temporal Databases." In *Proceedings of SIGMOD'04*, pp. 623–634, Paris, France, 2004. ACM, New York, USA.

[NCCar] Yanming Nie, R. Cocci, Zhao Cao, Yanlei Diao, and Prashant Shenoy. "SPIRE: Efficient Data Interpretation and Compression over RFID Streams." *IEEE Transactions on Knowledge and Data Engineering*, 2011 (to appear).

[NUY10] Takanobu Nakahara, Takeaki Uno, and Katsutoshi Yada. "Extracting Promising Sequential Patterns from RFID Data Using the LCM Sequence." In *Proceedings of the 14th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES'10)*, Cardiff, UK, 2010.

[OP07] Jan Ondrus and Yves Pigneur. "An Assessment of NFC for Future Mobile Payment Systems." In *Proceedings of the International Conference on the Management of Mobile Business*, Toronto, Canada, 7 2007.

[OTY09] Kyosuke Osaka, Tsuyoshi Takagi, Kenichi Yamazaki, and Osamu Takahashi. "An Efficient and Secure RFID Security Method with Ownership Transfer." In *RFID Security*, pp. 147–176. Springer US, 2009.

[PCC04] J. M. Patel, Y. Chen, and V. Chakka. "STRIPES: An Efficient Index for Predicted Trajectories." In *Proceedings of SIGMOD'04*, pp. 635–646, Paris, France, 2004. ACM, New York, USA.

[PSL11] Balaji Palanisamy, Aameek Singh, Ling Liu, , and Bhushan Jain. "Purlieus: Locality-aware Resource Allocation for MapReduce in a Cloud." In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, Seattle, WA, USA, November 2011.

[PTD07] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. "Service-Oriented Computing: State of the Art and Research Challenges." *IEEE Computer*, **40**(11):38–45, 2007.

[RC08] Damith C. Ranasinghe and Peter H. Cole. *Networked RFID Systems and Lightweight Cryptography: Raising Barriers to Product Counterfeiting*. Springer, 2008.

[RCG10] Michal Rosen-Zvi, Chaitanya Chemudugunta, Thomas Griffiths, Padhraic Smyth, and Mark Steyvers. "Learning Author-topic Models from Text Corpora." *ACM Transactions on Information Systems*, **28**:4:1–4:38, January 2010.

[RCT06] Melanie R. Rieback, Bruno Crispo, and Andrew S. Tanenbaum. "The Evolution of RFID Security." *IEEE Pervasive Computing*, **5**(1):62–69, March 2006.

[RDT09] George Roussos, Sastry S. Duri, and Craig W. Thompson. "RFID Meets the Internet." *IEEE Internet Computing*, **13**(1):11 –13, 1-2 2009.

[Rip01] M. Ripeanu. "Peer-to-Peer Architecture Case Study: Gnutella Network." In *International Conference on Peer-to-Peer Computing (P2P'01)*, Los Alamitos, CA, USA, 2001.

[RKB06] Ralf Rantzau, Karin Kailing, Steve Beier, and Tyrone Grandison. "Discovery Services - Enabling RFID Traceability in EPCglobal Networks." In *13th International Conference on Management of Data (COMAD'06)*, Delhi, India, 2006.

[RRH04] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, and Scott Shenker. "Brief Announcement: Prefix Hash Tree." In *Proceedings of PODC'04*, pp. 368–368, St. John's, Canada, 2004. ACM, New York, USA.

[RSZ10] Damith C. Ranasinghe, Quan Z. Sheng, and Sherali Zeadally. *Unique Radio Innovation for the 21st Century: Building Scalable and Global RFID Networks*. Springer, 2010.

[RWN07] C. Robson, Y. Watanabe, and M. Numao. "Parts Traceability for Manufacturers." In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*, Istanbul, Turkey, April 2007.

[SBE01] Sanjay Sarma, David Brock, and Daniel Engels. "Radio Frequency Identification and the Electronic Product Code." *IEEE Micro*, **21**(6):50–54, 11/12 2001.

[SLZ08a] Q. Z. Sheng, X. Li, and S. Zeadally. "Enabling Next-Generation RFID Applications: Solutions and Challenges." *IEEE Computer*, **41**(9):21–28, September 2008.

[SLZ08b] Quan Z. Sheng, Xue Li, and Sherali Zeadally. "Enabling Next-Generation RFID Applications: Solutions and Challenges." *IEEE Computer*, **41**(9):21–28, September 2008.

[SM07] C. Sutton and A. McCallum. "An Introduction to Conditional Random Fields for Relational Learning." In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[SPT04] J. Sun, D. Papadias, Y. Tao, and B. Liu. "Querying about the Past, the Present, and the Future in Spatio-Temporal Databases." In *Proceedings of ICDE'04*, pp. 202–211, Boston, USA, 2004. IEEE Computer Society, Washington, DC, USA.

[Sto01] I. Stoica et. al. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications." In *Proceedings of SIGCOMM'01*, San Diego, USA, 2001. ACM, New York, USA.

[SWC97] A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. "Modeling and Querying Moving Objects." In *Proceedings of ICDE'97*, pp. 422–432, Pennsylvania, USA, 1997. IEEE Computer Society, Washington, DC, USA.

[SWR10] Quan Z. Sheng, Yanbo Wu, and Damith C. Ranasinghe. "Enabling Scalable RFID Traceability Networks." In *Proceedings of the 24th International Conference on Advanced Information Networking and Applications*, Perth, Australia, 2010.

[THI10] Juan M. Tirado, Daniel Higuero, Florin Isaila, Jess Carretero, and Adriana Iamnitchi. "Affinity P2P: A Self-organizing Content-based Locality-aware Collaborative Peer-to-peer Network." *Computer Networks*, **54**(12):2056 – 2070, 2010.

[THS07] E. Tanin, A. Harwood, and H. Samet. "Using a Distributed Quadtree Index in Peer-to-Peer Networks." *The VLDB Journal*, **16**(2):165–178, 2007.

[TYJ09] K. Tzoumas, M. Yiu, and C. S. Jensen. "Workload-aware Indexing of Continuously Moving Objects." *PVLDB*, **2**(1):1186–1197, 2009.

[VGS05] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays." *Journal OF Network and Systems Management*, **13**(2):107–217, 2005.

[Vit85] Jeffrey S. Vitter. "Random Sampling with a Reservoir." *ACM Transactions on Mathematical Software*, **11**(1):37–57, 1985.

[VRC08] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. "A Break in the Clouds: Towards a Cloud Definition." *Computer Communication Review*, **39**:50–55, December 2008.

[Wan06] Roy Want. "An Introduction to RFID Technology." *IEEE Pervasive Computing*, **5**(1):25–33, January-March 2006.

[WFC05] Baohua Wei, Gilles Fedak, and Franck Cappello. "Scheduling Independent Tasks Sharing Large Data Distributed with BitTorrent." In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (GRID'05)*, Seattle, USA, 2005.

[WJO09] Sai Wu, Shouxu Jiang, Beng Chin Ooi, and Kian-Lee Tan. "Distributed Online Aggregations." *PVLDB*, **2**(1):443–454, 2009.

[WL05a] Fusheng Wang and Peiya Liu. "Temporal Management of RFID Data." In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, Trondheim, Norway, 2005.

[WL05b] Fusheng Wang and Peiya Liu. "Temporal Management of RFID Data." In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, Trondheim, Norway, September 2005.

[WLL10] Fusheng Wang, Shaorong Liu, and Peiya Liu. "A Temporal RFID Data Model for Querying Physical Objects." *Pervasive and Mobile Computing*, **6**(3):382–397, 2010.

[WLO08] Sai Wu, Jianzhong Li, Beng Chin Ooi, and Kian-Lee Tan. "Just-in-time Query Retrieval over Partially Indexed Data on Structured P2P Overlays." In *Proceedings of SIGMOD'08*, pp. 279–290, Vancouver, Canada, 2008. ACM, New York USA.

[WPS00] M. Wiesmann, F. Pedone, A. Schipe, B. Kemme, and G. Alonso. "Understanding Replication in Databases and Distributed Systems." In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, Taipei, Taiwan, April 2000.

[WRS11] Yanbo Wu, Damith C. Ranasinghe, Quan Z. Sheng, Sherali Zeadally, and Jian Yu. "RFID Enabled Traceability Networks: a Survey." *Distributed and Parallel Databases*, **29**(5-6):397–443, 2011.

[WSR11a] Yanbo Wu, Quan Z. Sheng, and Damith Ranasinghe. "Peer-to-Peer Objects Tracking in the Internet of Things." In *Proceedings of the 40th International Conference on Parallel Processing (ICPP'11)*, Taipei, Taiwan, 2011.

[WSR11b] Yanbo Wu, Quan Z. Sheng, and Damith C. Ranasinghe. "Tracing Moving OBjects in Internet-Based RFID Networks." In *Proceedings of the 25th International Conference on Advanced Information Networking and Applications Workshops*, Singapore, Singapore, 2011.

[XCZ11] Pengcheng Xiong, Yun Chi, Shenghuo Zhu, Hyun Jin Moon, Calton Pu, and H. Hacigumus. "Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment." In *Proceedings of the 27th International Conference on Data Engineering (ICDE 2011)*, Hannover, Germany, April 2011.

[YG01] Beverly Yang and Hector Garcia-Molina. "Comparing Hybrid Peer-to-Peer Systems." In *Proceedings of the 27th International Conference on Very Large Data Bases*, Roma, Italy, 2001.

[ZCB10] Qi Zhang, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges." *Journal of Internet Services and Applications*, **1**:7–18, 2010.

[ZCJ09] M. Zhang, S. Chen, C. S. Jensen, B. Ooi, and Z. Zhang. "Effectively Indexing Uncertain Moving Objects for Predictive Queries." *PVLDB*, **2**(1):1198–1209, 2009.

[ZFM10] Holger Ziekow, Benjamin Fabian, Cristian Mller, , and Oliver Gnther. "RFID in the Cloud: A Service for High-Speed Data Access in Distributed Value Chains." In *Proceedings of the 16th Americas Conference on Information Systems (AMCIS 2010)*, Lima, Peru, August 2010.

[ZKC04] Roger Zimmermann, Wei-Shinn Ku, and Wei-Cheng Chu. "Efficient Query Routing in Distributed Spatial Databases." In *Proceedings of GIS'04*, pp. 176–183, Washington DC, USA, 2004. ACM, New York, USA.

[ZLL09] Wen Zhao, Xueyang Liu, Xinpeng Li, Dianxing Liu, and Shikun Zhang. "Research on Hierarchical P2P Based RFID Code Resolution Network and Its Security." In *Proceedings of the Fourth International Conference on Frontier of Computer Science and Technology*, Shanghai, China, 2009.

[ZLZ09] Wen Zhao, Xueyang Liu, Shikun Zhang, Bingchen Chen, and Xinpeng Li. "Hierarchical P2P Based RFID Code Resolution Network: Structure, Tools and Application." In *Proceedigns of International Symposium on Computer Network and Multimedia Technology (CNMT '09)*, Wuhan, China, January 2009.

[ZWS10] S. Zhang, J. Wang, R. Shen, and J. Xu. "Towards Building Efficient Content-Based Publish/Subscribe Systems over Structured P2P Overlays." In *Proceedings of ICPP'10*, pp. 258–266. IEEE Computer Society, Washington, DC, USA, 2010.

[ZZP03] Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao, and Dik Lun Lee. "Location-based Spatial Queries." In *Proceedings of the 22th ACM Special Interest Group on Management of Data*, San Diego, USA, 2003.