Keio University

Master's Thesis

Robust and High Performance RFID Middleware with Entropy Filtering

Author:
Iori Mizutani

Chief Examiner:
Dr. Jin Mitsugi
Sub Examiners:
Dr. Jun Murai
Dr. Rodney D. Van Meter

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Media and Governance [M.M.G.]

in the

Graduate School of Media and Governance

January 12th, 2018

KEIO UNIVERSITY

Abstract

Graduate School of Media and Governance

Master of Media and Governance [M.M.G.]

Robust and High Performance RFID Middleware with Entropy Filtering

by Iori MIZUTANI

Radio-frequency identification (RFID) has become a key component in recent emerging industrial automation, and the middleware is now required to collect and properly distribute massive amount of RFID events, which is referred to as “filtering” in this thesis. I propose Entropy Filtering for building robust and high-performance RFID middleware for use in realistic situations. Multiple competing standards for RFID are vying for acceptance and stalling integration of RFID systems. For such surrounding issues, I first study the structure of IDs specified in the international standards and clarify how to ensure the interoperability between the multiple standards, which is necessary before combining RFID into existing inventory or logistics management systems.

I defined three distinct scenarios with the concrete criteria from performance evaluation of an existing implementation. For the evaluation, I developed an RFID reader emulator to produce a virtual population of tags to observe the behavior of the implementation. Entropy Filtering is a concept of filtering technique that collects and analyze the environmental factors such as event tra□cs and request types to automatically adapt the filtering tree topology to achieve fast filtering. I applied this to the filtering engine of RFID middleware with four algorithms optimized for the scenarios. The performance is evaluated with the emulator, and it is shown that the Entropy Filtering outperforms. Lastly, I point out non-trivial know-how that should be contained for the middleware deployment focusing on the parallelization of the middleware instances and load-balancing architecture. Another important characteristic of this work is the whole set of software is developed as open source code and this thesis provides detailed explanation of the reasoning behind each step to reproduce the experiments conducted. This thesis gives the reader all the means to deepen understanding of IDs in RFID as well as protocols and formats.

Acknowledgements

"I am the combined e□ort of everyone I've ever known."

In the very first place, I would like to thank my advisor Professor Jin Mitsugi for all the support and encouragement that he has provided in the last few years. Without him, I wouldn't be able to devote all my energy to work through, and I would have been nowhere near where I am standing right now. I would like to thank the Auto-ID fellows with whom I spent most of my time in SFC and Z202. I also want to thank Professor Jun Murai and Associate Professor Rodney D. Van Meter III for giving me the fruitful advice.

I thank Masafumi Nakane and Taisuke Sato for being my trustful seniors. I thank Evangelos "obot" Spyrou for constructive advice. I appreciate Kohei "*fox*" Yamamoto for sharing the last few weeks of growing impatience, frustration, depression, and the joy from the work seemed almost never-ending. I can't thank enough Rina "*enori*" Ueno for her attentiveness, not to mention, I enjoyed her curry last night. I thank Hiromu "*kamekame*" Kamei for being my steadfast junior, he often made me aware of something eye-opening. I thank Lisa "*alaine*" Obata for giving me the cheerful mood, I miss her from the lab these days. I thank Tomonao "*jintomo*" Jin for bringing in the calm feeling and a feeling of relief, I truly wish you will be back soon. I appreciate Mondo "*mondo*" Saito so much for visiting the lab during the break and encouraging me to focus on this thesis. I thank Nao "*naonanao*" Yamato for her e□ort, I've been surprised by how thoroughly she does everything and got straighten myself.

I thank all the MAUI members for pacing my project milestones, the friendly atmosphere has been always helpful.

I also thank KEIO EDGE project for funding me to challenge the new area of study in Berkeley.

I thank Florian Michahelles, Mareike Kritzler, Kimberly Garcia Garcia, Jack Hodges, and Dan Yu for welcoming me with warmth in the Siemens lab.

I thank Simon Mayer for inspiring me to think of the carrier in Phd. I thank Markus Funk for being such a good talker and listener, I learned a lot about how a researcher should be, and 420 Mason. I thank Fidencio Tamayo for being the best roommate of my life, his tireless energy and considerate words inspired me so much. I thank Yudai Yamagishi for the good time I spent in California, the beers and the oysters. I thank Koustubh Gaikwad for being my soul mate and always being there to chat with. I thank Rei Thomas for giving me courage to always challenge new things without hesitation. I thank Takumi Sakai for being a modest and respectable man. I thank Shunsuke Matsukuma for a mind of exploration.

Lastly, I want to thank my family, Kunihiko and Fumiko for their devoted love.

Contents

Abstract i

Acknowledgements ii

Contents iii

List of Figures vi

List of Tables viii

List of Algorithms ix

List of Abbreviations x

1 Introduction 1
1.1 Contributions . . . 3
1.2 Organization . . . 3

2 Multi-code RFID and Middleware 5
2.1 Problems of International Standardization in UHF-band RFID . . . 5
2.2 Coding Scheme . . . 7
2.2.1 Electronic Product Code (EPC) . . . 8
2.2.2 ISO . . . 11
2.2.3 uiigen – A command-line tool for binary encoding EPC/UII generator . . . 14
2.3 Components for RFID Middleware . . . 14
2.3.1 Subscription, Notification and Event Cycle . . . 14
2.3.2 Application Level Events (ALE) . . . 16
2.3.3 Low Level Reader Protocol (LLRP) . . . 17
2.3.4 Filtering Engine . . . 18
2.3.5 Related Commercial RFID Middleware Products . . . 19

3 Preliminary Experiment 22
3.1 Performance Measurement by RFID Reader Emulation . . . 22
3.1.1 golemu – An RFID reader emulator . . . 22
3.2 Observation Analysis . . . 25
3.2.1 Notification Delay . . . 25
3.2.2 Linearly Proportional Iteration and Variable Conversions . . . 27
3.2.3 Redundant Subscriptions . . . 27
3.2.4 Biased ID population . . . 28
3.3 Criteria for Robustness and High Performance . . . 29

4 Entropy Filtering 31

4.1 Parameter Assessment and Proposition . . . 31
4.2 Implementation of Entropy Filtering . . . 33
4.2.1 gosstrak-fc – A Filtering & Collection RFID middleware . . . 34
4.3 Filtering Algorithms . . . 35
4.3.1 Linear Search (List) . . . 35
4.3.2 Patricia Trie . . . 36
4.3.3 Optimal Binary Search Tree . . . 41
4.3.4 Splay Tree . . . 45
4.3.5 Filtering Execution . . . 46

5 Evaluation of the Filtering Engine 48
5.1 Scenario 1 – A container terminal . . . 48
5.2 Scenario 2 – A convenience store . . . 52
5.3 Scenario 3 – A shopping mall . . . 56
5.4 Integrity of ID Pattern Matching . . . 59

6 Techniques for Practical Deployment 62
6.1 Parallelization and Horizontal Scaling . . . 62
6.1.1 Local and Remote Parallel Filtering . . . 62
6.1.2 Analysis with Queuing Theory . . . 63
6.2 RFID Event Tra□c Load Balancing . . . 67
6.3 Subscription Report Aggregation . . . 67
6.4 Portability and Deployment Automation . . . 67

7 Conclusion 68
7.1 Recapitulation . . . 68
7.2 Future Work . . . 68

A Multi-code Binary Encoding Summary 70
A.1 EPC . . . 70
A.1.1 SGTIN-96 . . . 70
A.1.2 SSCC-96 . . . 70
A.1.3 GRAI-96 . . . 71
A.1.4 GIAI-96 . . . 71
A.2 ISO . . . 71
A.2.1 17363 . . . 71
A.2.2 17365 . . . 72

B uiigen – A command line tool to generate EPC/ISO standard conformable IDs 73
B.1 Installation and Synopsis . . . 73

C golemu – An RFID reader emulator 75
C.1 Installation and Synopsis . . . 75

D gosstrak-fc – An RFID middleware 77
D.1 Installation and Synopsis . . . 77
D.2 Implementation of the Selected Algorithms . . . 78
D.2.1 List (Linear Search) . . . 79
D.2.2 Patricia Trie . . . 81
D.2.3 Optimal Binary Search Tree (Optimal BST) . . . 86
D.2.4 Splay Tree . . . 90

D.2.5 Filter 92

Bibliography 96

List of Figures

2.1 Typical data capture architecture 6
2.2 Logical memory banks in the UHF-band RF tags 7
2.3 Stored Protocol Control bits assignment 8
2.4 EPC coding table example: SGTIN-96 9
2.5 Packaging in ISO/IEC 15418-x 11
2.6 Description of DIs for ISO/IEC 15459-1 12
2.7 Six-bit encoding for the 15434 direct encoding method 13
2.8 Binary encoding of ISO 17363 13
2.9 Subscription and Notification 15
2.10 LRSpec to register a logical reader in the middleware 16
2.11 ECSpec for subscribing ID filtering patterns with ECFilterSpec 17
2.12 ECReports to be notified from the middleware 18
2.13 Sequential diagram of LLRP communication 19
2.14 Binary encoding of RO_ACCESS_REPORT message 19
2.15 Binary encoding of TagReportData parameter 20
2.16 Binary encoding of EPCData parameter 20
2.17 Binary encoding of EPC-96 parameter 20
2.18 Structure of various filtering engines with a common interface 21

3.1 Overview of golemu's software design and interfaces 24
3.2 Overview of an RFID middleware and emulator setup 24
3.3 golemu running on Raspberry Pi 25
3.4 Java Mission Control profiling of Fosstrak with Java Flight Recorder . 25
3.5 The top two most sampled method calls per package in Fosstrak implementation 26
3.6 Delay propagation in-between event cycles 27
3.7 Redundant filters in subscriptions 28

4.1 A concept of dynamic feedback cycle 32
4.2 Overview of Entropy Filtering implementation for an RFID middleware 33
4.3 Sequential diagram of Entropy Filtering in an RFID middleware . . . 34
4.4 Patricia Trie of ID prefix filtering 37
4.5 Invalid node in Patricia Trie 37
4.6 Visualization of locality per node in Patricia Trie 40
4.7 Optimal Binary Search Tree (Optimal BST) of ID prefix filtering . . . 42
4.8 Visualization of locality per node in Optimal BST 44
4.9 Bit-Filter Patterns 46
4.10 Byte-Filter Patterns 47

5.1 Simulation Scenario 1: 148825 IDs / 721 Filters 49
5.2 Histogram and Kernel density estimation of the subscribed filters in Scenario 1 dataset 50

5.3 Performance result for Scenario 1 per filtering algorithm with di□erent dataset 51
5.4 Overview of Scenario 2 52
5.5 Simulation Scenario 2: 44456 IDs / 47561 Filters 53
5.6 Histogram and Kernel density estimation of the subscribed filters in Scenario 2 dataset 54
5.7 Performance result for Scenario 2 per filtering algorithm with di□erent dataset 55
5.8 Overview of Scenario 3 56
5.9 Simulation Scenario 3: 15915 IDs / 4763 Filters 57
5.10 Performance result for Scenario 3 per filtering algorithm with di□erent dataset 58
5.11 Non-prefix ID pattern example: any POS item 59
5.12 Non-prefix ID pattern example: any item of a Company Prefix 59
5.13 Filtering integrity in Venn diagram 60
5.14 An ideal filtering engine 61

6.1 Size of the filtering engine by the number of filters 63
6.2 M/M/m State Space Diagram 64

List of Tables

2.1 EPC headers 9
2.2 SGTIN Partition Value 10
2.3 Filter Value 10
2.4 SGTIN-96 binary encoding example 10
2.5 DI mapping for GS1 AI 12
2.6 Layer of supply chain in 1736x 12
2.7 Application Family Identifier for 1736x 13

4.1 The weight values for the five filters in the example. 41
4.2 The effect of byte-wise comparison to eliminate string comparison process. 47

6.1 Symbols in the M/M/m State Space Diagram. 64

List of Algorithms

4.3.1 Linear Search 35
4.3.2 Patricia Trie: Main 38
4.3.3 Patricia Trie: Functions 39
4.3.4 Optimal Binary Search Tree 43
4.3.5 Splay Tree (snippet) 45

List of Abbreviations

AFI	Application Family Identifier (ISO/IEC FDIS, 2010)
AI	Application Identifier (GS1, 2017b)
ALE	Application Level Events (GS1, 2009; ISO/IEC, 2011)
BIC	Bureau International des Containers et du Transport Intermodal[1]
C1G2	Class 1 Generation 2 (GS1, 2015)
CIN	Company Identification Number
DI	Data Identifier (ISO/IEC, 2009a)
DSFID	Data Storage Format IDentifier (GS1, 2015; ISO/IEC, 2013b)
EC	Event Cycle (GS1, 2009)
EI	Equipment Identifier (ISO, 1995)
EPC	Electric Product Code (Mealling, 2008)
F&C	Filtering and Collection (GS1, 2014)
GIAI	Global Indivisual Asset Identifier
GRAI	Global Returnable Asset Identifier
IAC	International Agency Code (ISO/IEC, 2014a)
ISO	International Organization for Standard (Goodwin and Apel, 2008)
LLRP	Low Level Reader Protocol (GS1, 2010; ISO/IEC, 2012)
NSI	Numbering System Identifier (GS1, 2015; ISO/IEC, 2013b)
OC	Owner Code (ISO, 1995)
OID	Object IDentifier (Howes et al., 1995; Mealling, 2000; Mealling, 2001)
POS	Point Of Sales
PC	Protocol Control (GS1, 2015; ISO/IEC, 2013b)
RFID	Radio Frequency IDentifer
SSCC	Serial Shipping Container Code
SGTIN	Serialized Global Trade Item Number
TDS	Tag Data Standard (GS1, 2017a)
TDT	Tag Data Translation (GS1, 2011)
URN	Uniform Resource Name (Moats, 1997; Daigle et al., 2002; Saint-Andre and Klensin, 2017)
WSDL	Web Service Description Language (W3C, 2007)
XSD	XML Schema Definition (W3C, 2012)

[1] https://www.bic-code.org/bic-codes/

Dedicated to my beloved gramma, Yuriko Iwasaki.

Chapter 1

Introduction

In the last decade, Radio Frequency Identifier (RFID) has played an important role in industries. The applications of RFID brought in operational e□ciencies improvement especially in the retailing industries and it is quickly accelerating these few years. RF tags attached to clothing in our local department stores have been used for shoplifting prevention (e.g.,Mighty Cube Co.,Ltd., 2017; Roberti, 2017). With enhancement of various Electronic Article Surveillance (EAS) tags with alarm triggers or ink spread functions (e.g., Catalyst Direct, 2017; Checkpoint Systems, Inc., 2016) on the market, retailers attained a great hedge against the risk of loss from retail thefts, not only by the system itself but also showing the public that the retail stores can realize the safety backed by a concrete technology without manpower. Since RF tags are designed to identify individual items, it is quite natural to see RFID integration in supply chain management (SCM) system or logistics operations.

In the late 2000s, the adoption of RFID was slow due to high cost for investment and technical di□culties (Paul, 2007; Computer Economics, Inc., 2007; Roberti, 2015). Even though some retail giants tried to deploy the RFID-based SCM system, they were not quite successful to turn up their suppliers to follow on a large scale (Gaudin, 2008). One of the major reasons that have slowed down the RFID integration is that there was no attractive enough incentive for suppliers to bring in RFID-based systems into their flows of operation. Individual item traceability seemed beneficial at a first glance, but the cost of that kind of implementation doesn't compensate with those unseen improvements. Most of the case in a large company, the R&D department, who leads RFID integration, is separated from the marketing and sales departments, who could utilize the traceability data and increase the revenue. From a di□erent point of view, the standardization of coding schemes in RF tags was another problem. When inter-operating among other vendors and suppliers, the use of standardized protocols is mandatory. There are two di□erent but major international standards: International Organization for Standardization (ISO) and GS1, which could not be easily co-operable.

Through the challenging period as such, the retail industries gradually found their ways to adopt RFID technologies in recent years (Roberti, 2017). Fashion apparel manufactures and garment rental companies are now starting to utilize washable RF tags to insert the tags into garments and high-fashion items without using a pouch or heat-seal tape (FUJITSU FRONTECH LIMITED, 2017). In the airline industry, RF tags is attached to aircraft parts to reduce the cost of maintenance and operation (International Air Transport Association (IATA), 2017). The concepts of IoT or Industrie 4.0 have become the top priority for many companies and RFID is captivating in the field of automation (Hermann, Pentek, and Otto, 2016).

Many retailers are stepping into Omni-Channel Retailing, which is defined to orchestrate the physical stores and online stores and unite their sales and logistics channel reaching customers. An interesting e□ort by Tyco Retail Solutions, 2016 with Macy's, the largest department store chain in the US, is called "Pick to the Last Unit" program. Typically, the number of available items are not accurately exposed to online store because they are not confident in the exact number of items in the enterprise-level inventory. That has forced them about 15% to 20% of inventory to be accounted for by the last unit in the store, and those massive amount of budget and unsold items have been at stake. The item-level RFID inventory bolsters Omni-channel fulfilment in both supplies and customer demands. Along with the new movements in those RFID integration, the other factors are also taking an upturn toward the acceleration of RFID use cases. In Japan, the price range for RF tags is finally mile-stoned to decrease from 10 JPY to 1 JPY (Dai Nippon Printing Co., Ltd., 2017). The cost reduction has become a boost for a new trend in the retail industries; automated check-out. Automated check-out is in fact a huge shift at the moment, and RF tags are playing the principal part to identify the purchasing items in the store (TOSHIBA TEC CORPORATION, 2017). Especially in Japan, the aging population and changing demographics are serious social issues. As a consequence, retail outlets are experiencing labor shortage, especially in convenience stores. There are more than 56,000 convenience stores in Japan, and they have become inseparable from our daily lives. Seven & i Holdings Co., Ltd., 2015 reported an average store handles 2,800 products, and 70% of them are replaced yearly. To encompass the significant role of convenience stores in the social background, The Ministry of Economy, Trade and Industry (METI), 2017 reached an agreement with five major convenience store chains to introduce RF tags for all the items in their convenience stores by 2025.

Likewise in logistics industries, RFID is a key to enable next generation optimization. For example, round-use of freight containers is gathering attentions in the field of international logistics. According to Japan Institute of Logistics Systems, 2013, up to 87% of international containers in transit are empty. The inland drayage of general freight containers generally includes empty containers in and out from container terminal at harbors. The conveyance of empty containers occurs after devanning items from the containers, and before loading items to the containers in remote depots. That results in ine□cient use of human and natural resources. One of the major reasons for this kind of ine□ciency comes from horizontal specialization of large freight transportation. In general, vertical integration of logistics is only possible for large-scale corporation, and there is no platform for small businesses to interoperate with other companies in logistics. Reducing those empty container conveyances is the first step of the cooperation by all the logistics operators. Making matches among shipping demands by di□erent companies (Container Matching) or reusing containers internally without returning them to harbors (Container Round-Use), will then reduce environmental burdens caused from heavy transportation near the terminal and also alleviate tra□c jams around depots, which positively a□ects drivers' labor condition by improving under-sta□ng problems.

Aside from container round-use, pallet vendors are advancing in introducing RFID to their services (NTT Electronics, 2014; Power Pallet Recycling Center, 2016). Pallets are often categorized as returnable assets and they are intended to be recycled from the first place. To realize visualization and orchestration of pallet management system and logistics, RF tags is the most suitable identifier media, and they can even provide the freight aggregation service on top of the pallet leasing business. Freight operators manage items by scanning attached RF tags at the terminal, and under which

circumstances most likely IDs of different international standard cohabit in a single location. Those different international coding scheme of IDs (called "multi-code") are problematic when it comes to the operation beyond vendors.

In any of the preceding examples, RFID middleware is inevitable. RFID middleware receives RFID events from RFID readers and bundle the report for appropriate destinations. For Electronic Product Code (EPC) RFID systems, there are several vendors producing RFID middleware (see Section 2.3.5). However, none of them are advertised as multi-code compatible and apparently we can't retrieve their performance evaluation, either. So far it has been acceptable to have an RFID middleware with *modest* performance, because there's no performance evaluation method like an RFID reader emulator. As many factors suggesting, we can anticipate unprecedentedly large traffic of RFID events in the industry regardless of sectors, and the genuine performance will become insufficient at some point.

1.1 Contributions

This thesis presents a robust RFID middleware to primarily filter IDs receiving from RFID readers at high performance for major use cases. In the next list I present some contributions of this work:

- Establishment of multiple international standard coding scheme organization
  The decisions made for which standard to use depend on the field of industry or even from companies to companies. The interoperability counts on the shared vocabulary based on the standardized platform; however, it is also challenging for each vendor to study a pile of specifications. I organized the scattered information in the early chapters for readers to understand the necessary specifications to generate codes for UHF RF tags and how we should interpret the conventions.
- Entropy Filtering The key concept Entropy Filtering is introduced to design a robust and high-performance RFID middleware. It enables environmental adaptive tuning for different use cases by giving feedback information to the middleware.
- RFID reader emulator for RFID system evaluation There has been no means to evaluate an RFID-based system with test cases before the production-ready deployment at the real site. I developed an RFID emulator as an instrumental tool for such performance assessment.
- Implementation of an RFID middleware The current and only open-source RFID middleware has been Fosstrak. Fosstrak is the only accessible RFID middleware at the moment, but their implementation is obsolete and not quite maintained these days. When dealing with a massive order of IDs and subscribed patterns to make matches, there needs to be a proper mechanism to maintain the functionality in a robust manner. I propose a new implementation of RFID middleware based on the state-of-art technology and attempt to elaborate the important aspects for building an RFID middleware.

1.2 Organization

The thesis is organized as follows.

- Chapter 2 establishes the organization of Multi-code schemes by revealing the difficulties to understand them, and elaborates the middleware role in the whole structure.
- Chapter 3 states the preliminary experiments and evaluation on a former prototype RFID middleware with an RFID reader emulator, then clarify the criteria as well as the scenarios to validate them.
- Chapter 4 proposes Entropy Filtering for the filtering engine of an RFID middleware with four different filtering algorithms based on the analysis of the criteria and scenarios.
- Chapter 5 evaluates the presented Entropy Filtering under the assumption from the scenarios.
- Chapter 6 describes supplementary suggestions on deployment regarding horizontal scaling parallelization and load-balancing.
- Chapter 7 concludes the thesis by recapitulating the core contributions and further observations with messages to practitioners.

Chapter 2

Multi-code RFID and Middleware

RFID is an automatic identification technology enabled by radio waves or electromagnetic fields. An RF tag is a microchip which receives the energy from a reader (interrogator) through the attached antenna, and responds with the data stored in the circuit. There are active RF tags with batteries and passive tags without batteries. RF tags are further classified by the frequency band they are used in.

- Low Frequency (LF) 125 KHz or 134 KHz, used for animal-tracking (ISO, 2011) and proximity card (ISO/IEC, 2009b).
- High Frequency (HF) 13.56 MHz, used for vicinity card (ISO/IEC, 2010a), payment, access control by Near Field Communication (NFC, ISO/IEC, 2013c; ECMA, 2013), MIFARE (ISO/IEC, 2016), and FeliCa (JIS, 2005) specifications.
- Ultra-High Frequency (UHF) 433 MHz and 860-960 MHz, used for item tagging and active *beacon* with batteries
- Microwave 2.45 GHz, used for road-tolling, mostly active RFID (ISO/IEC, 2015). The 5.8 GHz (ISO/IEC18000-5) has been withdrawn due to the lack of use.

The IDs discussed throughout this thesis are mainly for UHF-band RFID intended for global supply chain management (SCM). The UHF frequency band is regulated by both EPC UHF Gen2 Air Interface Protocol (C1G2, GS1, 2015) and ISO/IEC 18000-6 Type C (ISO/IEC, 2013b). Unlike the specifications in the physical layer, the coding schemes varies by each standard family (hereinafter, referred to as multi-code). The UHF-band RFID is intended to operate with the existing SCM systems such as barcode or two-dimensional barcode. FIGURE 2.1 shows how the RFID should be integrated in and interoperate with a system using barcode as enterprise SCM. To deal with the multi-code environment, this chapter clarifies what component is defined by which standard, and the difference as well as the similarity between the two international standards.

2.1 Problems of International Standardization in UHF-band RFID

With the emerging background mentioned in Chapter 1, RFID integration is becoming more and more common in industry. Yet, the IDs carried by RF tags are often

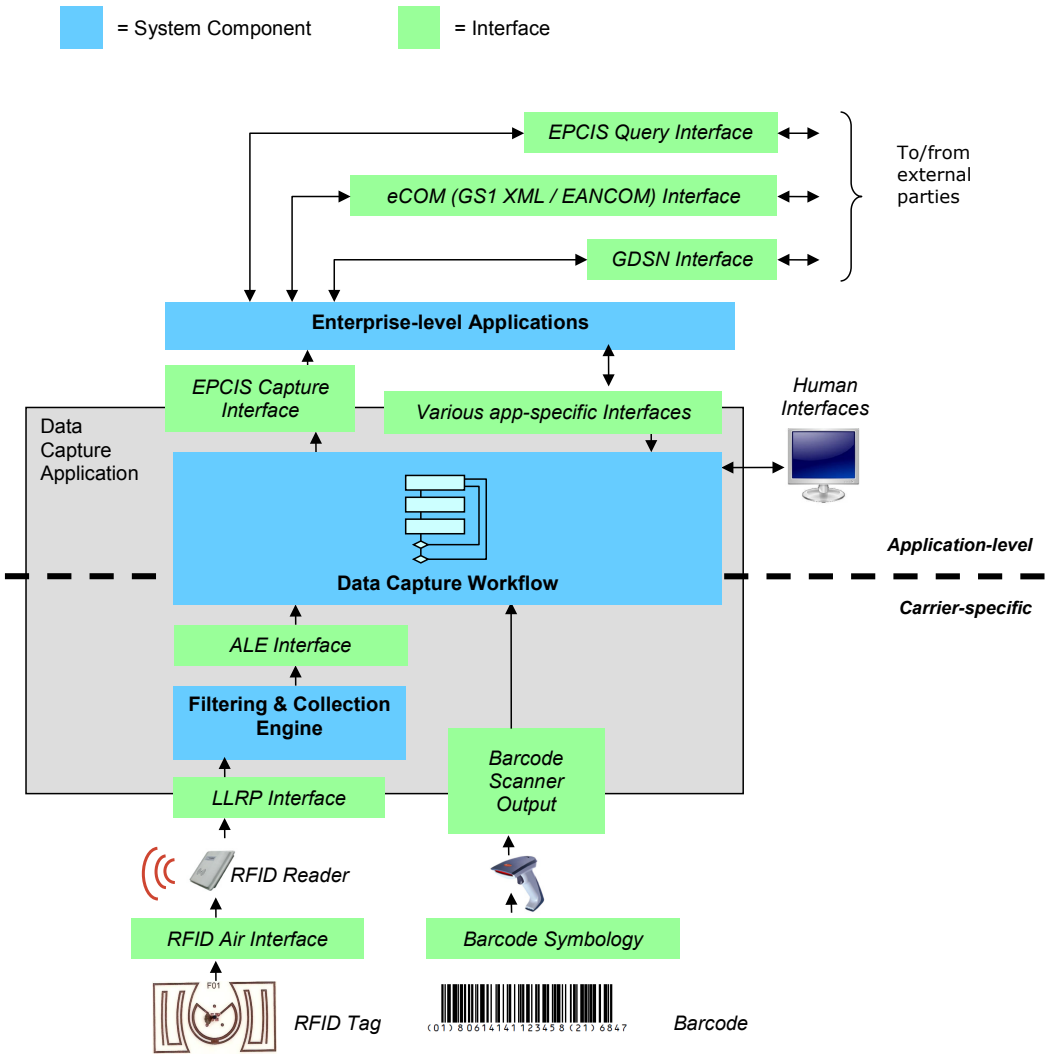


FIGURE 2.1: Typical data capture architecture from GS1 RFID/Barcode Interoperability Guideline (GS1, 2016).

not in conformity with the international standard; some are just invalid with erroneous encoding and the others are even using their own proprietary format. Such deviancy is critical for an operation like global logistics, and it prevents the whole ecosystem from working normally. Not only the legitimacy, the interoperability in the multi-code environment is another issue. Fostering an international consensus has to be dealt with immediately, but it has been sluggish like other struggle for supremacy or issues in vertically divided structure. Which standard, GS1 or ISO, to be used completely depends on the field of business or even a company. Inside their information management structure, usually the system is rooted only in either standard and making it more di□cult to adopt the other one as the migration cost is too high to realize the multi-code integration.

One of the major reasons why it is so di□cult is that the concrete information is not well organized to be understood thoroughly. When companies and subordinating vendors plan to implement an UHF-band RFID system along with legitimate standardization, they most likely need consultants who know very well about the

complexity of both the standards. In particular, the ISO standards are not accessible to anyone unless purchased, so we do not precisely know the contents before reading them. What is even worse, the necessary information is dispersed among di□erent numbering. In the following sections, I attempt to organize the confusing structure in ISO standards and relate them to GS1's.

2.2 Coding Scheme

UHF-band RF tags of C1G2 or ISO/IEC18000-6 Type C stores data in the memory banks as indicated in FIGURE 2.2.

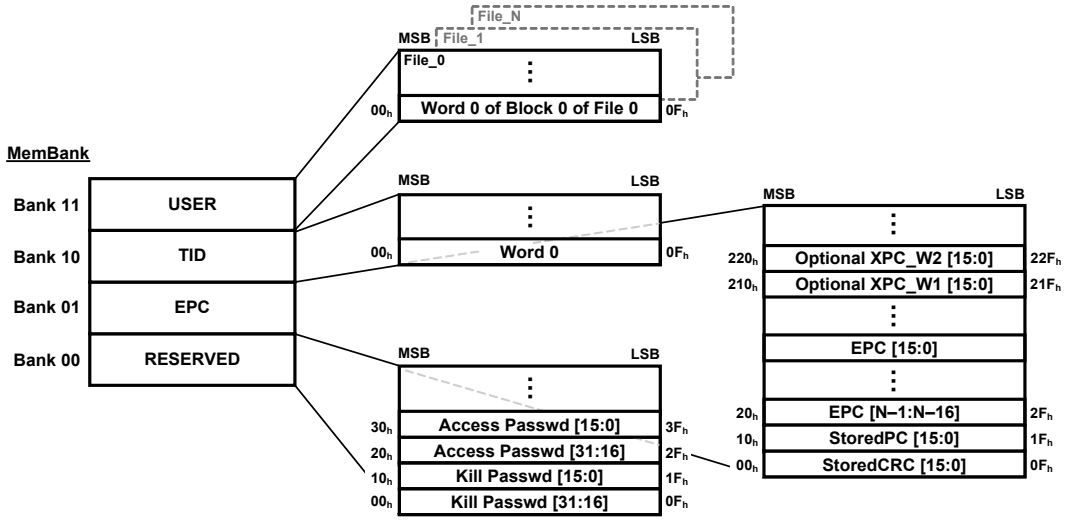


FIGURE 2.2: Logical memory banks in the UHF-band RF tags from EPC UHF Gen2 Air Interface Protocol (GS1, 2015).

The both standards describe the conformable air protocol and commands for interrogation and the logical map of the memory allows the stored data to be accessed in an analogous way. However, it is a di□erent story for the stored data. GS1 and ISO each has its own format and encoding rules to represent the ID, and we need to look for further materials to fully understand the formality of the IDs.

Use of Protocol Control bits (PC) is the most reasonable way to distinguish an RF tag from GS1 to ISO standard ID. PC is stored just before the segment used for IDs (10_h-$1F_h$ of Bank 01 in FIGURE 2.2), and it is a 2-byte data contains the length of ID in words (=16 bits) at 10_h-14_h, a toggle bit called User Memory Indicator at 15_h to indicate if the tag has data in Bank 11, a toggle bit to indicate if the tag has extended PC at 16_h, a toggle bit to indicate the ID is of whether GS1 or non-GS1 family at 17_h, and supplementary meta data for ISO IDs called Application Family Identifier at 18_h-$1F_h$.

Once we know of which standard the RF tag is, we decode the ID by each encoding rule. The binary encoding of the IDs related to this study is displayed in appendix A in addition to the examples in the following clauses.

StoredPC bit assignments

Application	MSB														LSB
	10_h	11_h	12_h	13_h	14_h	15_h	16_h	17_h	18_h	19_h	$1A_h$	$1B_h$	$1C_h$	$1D_h$	$1E_h$	$1F_h$
GS1 EPCglobal	**L4**	**L3**	**L2**	**L1**	**L0**	**UMI**	**XI**	**T**=0	RFU							
Non-GS1 EPCglobal	**L4**	**L3**	**L2**	**L1**	**L0**	**UMI**	**XI**	**T**=1	AFI as defined in ISO/IEC 15961							

FIGURE 2.3: Stored Protocol Control bits assignment from EPC UHF Gen2 Air Interface Protocol (GS1, 2015).

2.2.1 Electronic Product Code (EPC)

EPC is a collective term for the identification code standardized by GS1, 2017b. The foundation of EPC is a generic product identification code GTIN and its family, and it is intended to be used for electronic tags including RF tags. With the consistency to existing barcode-based system, it can be widely used for any electronic tag systems. An example of an EPC is SGTIN, which is a unique ID for individual items and consists of GTIN and Serial numbers. In GS1 coding scheme, there are seven representations for a single EPC. The most platform-agnostic format of an EPC is Pure Identity and it is expressed in Universal Resource Name (URN, Moats, 1997) for human readability and compatibility between heterogeneous systems. For instance,

`urn:epc:id:sgtin:458960468.0022.86`

is a Pure Identity of SGTIN with GS1 Company Prefix “458960468”, Item Reference “22”, and serial “86”. Meanwhile, an EPC stored in RF tags is in binary format. The binary encoding procedure in defined in EPC Tag Data Standard (TDS, GS1, 2017a). The general structure of an EPC in binary encoding consists of a fixed length header followed by a series of respective fields to the header value. (TABLE 2.1)

Some EPCs, whose usage with RFID is not practical or not clearly defined in conjunction with other standard, have this EPC headers. For example, a GTIN plus a Batch/Lot (LGTIN) doesn’t have neither the corresponding EPC header or the coding table for binary encoding. At the moment, we cannot use those EPC on RF tags. For an EPC with the EPC header defined, there is a coding table defined in TDS. SGTIN-96 as an example, FIGURE 2.4 is the coding table, TABLE 2.2 is the partition table, and TABLE 2.3. To issue an valid binary encoded SGTIN-96, the binary should satisfy this encoding rule; the EPC header, filter value, partition value, GS1 Company Prefix, Item Reference, and serial of the predetermined length in the right order.

For example, when translating the previous SGTIN Pure Identity example into the binary encoding format of SGTIN-96, each field is filled as shown in TABLE 2.4

TABLE 2.1: EPC headers snippet from TDS (GS1, 2017a).

Header Value (binary)	Header Value (hex)	Encoding Length (bits)	Coding Scheme
0010 1100	2C	96	GDTI-96
0010 1101	2D	96	GSRN-96
0010 1110	2E	96	GSRNP
0010 1111	2F	96	USDoD-96
0011 0000	30	96	SGTIN-96
0011 0001	31	96	SSCC-96
0011 0010	32	96	SGLN-96
0011 0011	33	96	GRAI-96
0011 0100	34	96	GIAI-96
0011 0101	35	96	GID-96
0011 0110	36	198	SGTIN-198
0011 0111	37	170	GRAI-170
0011 1000	38	202	GIAI-202
0011 1001	39	195	SGLN-195
0011 1100	3C	96	CPI-96
0011 1110	3E	174	GDTI-174
0011 1111	3F	96	SGCN-96
0100 0000	40	110	ITIP-110
0100 0001	41	212	ITIP-212

	SGTIN-96					
URI Template	urn:epc:tag:sgtin-96:*F.C.I.S*					
Total Bits	96					
Logical Segment	EPC Header	Filter	Partition	GS1 Company Prefix (*)	Indicator (**) / Item Reference	Serial
Logical Segment Bit Count	8	3	3	20-40	24-4	38
Coding Segment	EPC Header	Filter	GTIN			Serial
URI portion		*F*	*C.I*			*S*
Coding Segment Bit Count	8	3	47			38
Bit Position	$b_{95}b_{94}...b_{88}$	$b_{87}b_{86}b_{85}$	$b_{84}b_{83}...b_{38}$			$b_{37}b_{36}...b_0$
Coding Method	00110000	Integer	Partition			Integer

FIGURE 2.4: EPC coding table example: SGTIN-96 from TDS (GS1, 2017a).

TABLE 2.2: SGTIN Partition Table from TDS (GS1, 2017a).

Partition Value (P)	GS1 Company Prefix		Item Reference	
	Bits (M)	Digits (L)	Bits (N)	Digits
0	40	12	4	1
1	37	11	7	2
2	34	10	10	3
3	30	9	14	4
4	27	8	17	5
5	24	7	20	6
6	20	6	24	7

TABLE 2.3: Filter Value Table from TDS (GS1, 2017a).

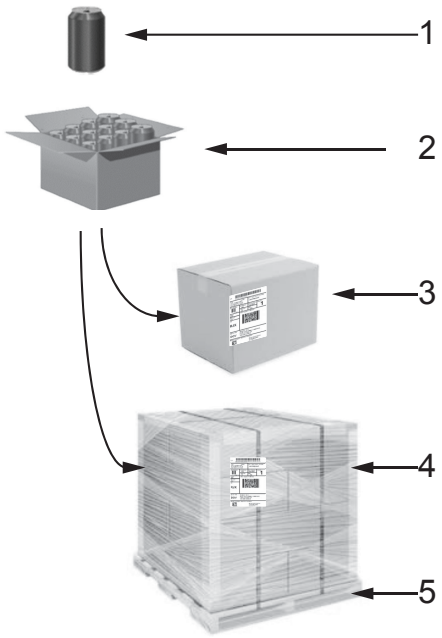
Type	Filter Value	Binary Value
All Others	0	000
Point of Sale (POS) Trade Item	1	001
Full Case for Transport	2	010
Inner Pack Trade Item Grouping for Handling	4	100
Unit Load	6	110
Unit inside Trade Item or component inside a product not intended for individual sale	7	111

TABLE 2.4: SGTIN-96 binary encoding example for urn:epc:id:sgtin:458960468.0022.86.

Field	Value (dec)	Value (bin)	Length (bits)	
EPC Header	48	00110000	8	
Filter Value	1	001	3	POS item
Partition Value	3	011	3	As the Company Prefix has 9 digits
Company Prefix	458960468	011011010110110 010111001010100	30	
Item Reference	22	00000000010110	14	The length is by Partition
Serial	86	00000000000000000000 00000000001010110	38	Fixed length

2.2.2 ISO

ISO IDs used in UHF-band RF tags (ISO/IEC 18000-6 Type C), are defined in ISO/IEC 15459-1 for Individual transport unit (ISO/IEC, 2014a), ISO/IEC 15459-4 for Individual products and product packages (ISO/IEC, 2014b), ISO/IEC 15459-5 for Individual returnable transport items (RTIs) (ISO/IEC, 2014c), and ISO/IEC 15459-6 for Groupings (ISO/IEC, 2014d). Each of them is classified into the four layers of the supply chain, from layer 1 (products) up to layer 4 (returnable transport items) as shown in FIGURE 2.5.



Key

1 primary packaging – consumer packaging – (product)
2 secondary packaging – outer packaging – (product package)
3 tertiary packaging – transport packaging – (transport unit)
4 tertiary packaging – unitized transport packaging – (transport unit)
5 pallet – (returnable transport item – RTI)

FIGURE 2.5: Packaging in ISO/IEC 15459-x from ISO 17365 (ISO, 2013c).

Practically, ISO/IEC 15459-1 is the most common IDs for automatic identification and data capture (AIDC) system; for example, World Customs Organization, 2004 recommends the use of ISO/IEC 15459-1 is the preferred option for the implementation of Unique Consignment Reference (UCR). UCR is a reference number to be reported during a Customs procedure for the reconciliation of international goods movements. ISO/IEC 15459-1 consists of Data Identifier (DI) defined in ISO/IEC 15418, also known as ASC MH10 (ISO/IEC, 2009a), and its corresponding structure of unique identifier for transport units. DI actually makes this standard also conformable with EPC. FIGURE 2.6 shows the description of DIs assigned for ISO/IEC 15459-1, and TABLE 2.5 is the DI mapping for GS1's Application Identifier (AI), which indicates the type of GS1 data content.

The encoding method of those ISO IDs to store in the memory bank of RF tags is defined by each layer of supply chain (TABLE 2.6).

In Bank 01 of the tag memory, Application Family Identifier (AFI) is required to be stored in addition to the fields needed for EPC. TABLE 2.7 shows the list of AFI defined for ISO 1736x.

Dissimilar to EPC, an ISO ID for ISO/IEC 18000-6 Type C has a variable length, and it follows six-bit encoding for ISO/IEC 15434 direct encoding method (ISO/IEC,

CATEGORY 2:	**Container Information**	
an2+an11	**7B**	Container serial number According to ISO 6346. OC EI CSN CD, where the OC is the three letter owner code assigned in cooperation with BIC, the EI is the one letter equipment category identifier, the CSN is 6-digit unique container identification assigned by the equipment owner, and CD is a modulus 11 check digit calculated in accordance with Annex A, ISO 6346
CATEGORY 10:	**License Plate**	
an1+an1...35	**J**	Unique license plate number Note 6
an2+an1...35	**1J**	Unique license plate number* assigned to a transport unit which is the lowest level of packaging, the unbreakable unit
an2+an1...35	**2J**	Unique license plate number* assigned to a transport unit which contains multiple packages
an2+an1...35	**3J**	Unique license plate number* assigned to a transport unit which is the lowest level of packaging, the unbreakable unit and which has EDI data associated with the unit
an2+an1...35	**4J**	Unique license plate number* assigned to a transport unit which contains multiple packages and which is associated with EDI data
an2+an1...20	**5J**	Unique license plate number* assigned to a mixed transport unit containing unlike items on a single customer transaction and may or may not have associated EDI data.
an2+an1...20	**6J**	Unique license plat number* assigned to a master transport unit containing like items on a single customer transaction and may or may not have associated EDI data.
	7J	Vehicle Registration License Plate Number (not unique without identification of country and issuing governmental region/authority)7

FIGURE 2.6: Description of DIs for ISO/IEC 15459-1 from ISO/IEC 15418 (ASC MH10, ISO/IEC, 2009a).

TABLE 2.5: DI mapping for GS1 AI snippet from ISO/IEC 15418 (ASC MH10, ISO/IEC, 2009a).

AI	Data Content	Format
00	Serial Shipping Container Code (SSCC)	J, 1J, 2J, 3J, 4J, 8S
01	Global Trade Item Number (GTIN or SCC-14)	8P
10	Batch or Lot Number	1T
30	Count of Items (Variable Measure Trade Item)	Q
+400	Customer's Purchase Order Number	K
402	Global Shipment Identification Number	2K
422	Country of Origin of a Trade Item	4L
8003	Global Returnable Asset Identifier	25B
8004	Global Individual Asset Identifier	1B, 5B

TABLE 2.6: Layer of supply chain in 1736x.

Layer	Standard
Freight containers	ISO 17363 (ISO, 2013a)
Returnable transport items (RTIs) and returnable packaging items (RPIs)	ISO 17364 (ISO, 2013b)
Transport units	ISO 17365 (ISO, 2013c)
Product packaging	ISO 17366 (ISO, 2013d)
Product tagging	ISO 17367 (ISO, 2013e)

2006) as shown in FIGURE 2.7.

TABLE 2.7: Application Family Identifier for 1736x.

AFI	Assigned organization or function
0xA1	ISO 17367 product tagging
0xA2	ISO 17365 transport unit
0xA3	ISO 17364 returnable transport unit
0xA4	ISO 17367 product tagging, but for hazardous materials
0xA5	ISO 17366 product packaging
0xA6	ISO 17366 product packaging, but for hazardous materials
0xA7	ISO 17365 transport unit, but for hazardous materials
0xA8	ISO 17364 returnable transport unit, but for hazardous materials
0xA9	ISO 17363 freight containers
0xAA	ISO 17363 freight containers, but for hazardous materials

Space	100000	0	110000	@	000000	P	010000
<EOT>	100001	1	110001	A	000001	Q	010001
“	100010	2	110010	B	000010	R	010010
<FS>	100011	3	110011	C	000011	S	010011
<US>	100100	4	110100	D	000100	T	010100
%	100101	5	110101	E	000101	U	010101
&	100110	6	110110	F	000110	V	010110
‘	100111	7	110111	G	000111	W	010111
(	101000	8	111000	H	001000	X	011000
)	101001	9	111001	I	001001	Y	011001
*	101010	:	111010	J	001010	Z	011010
+	101011	;	111011	K	001011	[	011011
,	101100	<	111100	L	001100	\	011100
-	101101	=	111101	M	001101	]	011101
.	101110	>	111110	N	001110	<GS>	011110
/	101111	?	111111	O	001111	<RS>	011111

FIGURE 2.7: Six-bit encoding for the 15434 direct encoding method from ISO/IEC 15962 (ISO/IEC, 2013a).

By taking an example of an ID for ISO 17363, FIGURE 2.8 shows the binary encoding of an ID “7BCSQU3054383”: “7B” as DI, “CSQ” as owner code, “U” as equipment category identifier, “305438” as serial, and “3” as check digit. The owner code and the equipment category are defined in ISO 6346 (ISO, 1995).

Offset	0	1	2	3	4	5	6
Character	7	B	C	S	Q	U	3
Binary	**110111**	**000010**	**000011**	**010011**	**010001**	**010101**	**110011**
Offset	7	8	9	10	11	12	13
Character	0	5	4	3	8	3	
Binary	**110000**	**110101**	**110100**	**110011**	**111000**	**110011**	**10**

FIGURE 2.8: Binary encoding of ISO 17363 for “7BCSQU3054383”.

Note that there is remaining “10” floating outside the original ID. The end of the encoding is defined by the 6-bit encoding rules in ISO/IEC 15962 (ISO/IEC, 2013a).

According to the rule, if the tag architecture supports an encoding unit of more than an 8-bit byte (e.g., ISO/IEC 18000-6 Type C uses a 16-bit word) then of the pad string “100000” is repeated as necessary in full and then truncated to fill the encoding space. This pad string maintains the byte-wise consistency of the encoded data to make the length to a multiple of 16-bit. Using the example of encoding on a 16-bit word structure, here are the eight end conditions:

- Exactly on the boundary — no action
- Requiring two pad bits — encode 10
- Requiring four pad bits — encode 1000
- Requiring six pad bits — encode 100000
- Requiring eight pad bits — encode 10000010
- Requiring ten pad bits — encode 1000001000
- Requiring twelve pad bits — encode 100000100000
- Requiring fourteen pad bits — encode 10000010000010

For the interchangeability between EPC, the use of Pure Identity is encouraged by some guidelines to represent an ISO ID; however, the entire element of the original ID cannot be expressed when using ISO/IEC 15961-1 (ISO/IEC, 2010b) or ISO/IEC 15434 data format of Object Identifier (OID) (Mealling, 2001). This is because in RFC1778 (Howes et al., 1995) it is defined that the syntax after < *descr* > field only allows numeric string. Hence, those ISO IDs which has characters cannot be translated into URN format with OID namespace. For this conflict, an accommodation of another namespace in URN for ISO IDs (e.g, urn:xoid or urn:epc:id:iso) could be a solution.

2.2.3 uiigen – A command-line tool for binary encoding EPC/UII generator

In the following chapters, a stack of IDs is needed to conduct experiments and simulation for multi-code RF tags. Generation of *random* IDs by following the proper encoding rules and each standard scheme is cumbersome as we went through the previous subsections. Therefore I made a command-line tool uiigen to create an arbitrary ID of the given standard coding scheme. The detailed usage is explained in appendix B. All the IDs used for later evaluation are generated with uiigen.

2.3 Components for RFID Middleware

The RFID middleware has two communication channels, one for subscribers and the other for interrogators. This section explains the role of each component in the context of RFID information system with nomenclature.

2.3.1 Subscription, Notification and Event Cycle

Similar to other message-oriented middleware system, UHF-band RFID system employs the Publish-subscribe model (Birman and Joseph, 1987). There are specific terms used to reference the RFID event subscription model.

- Subscriber (Event Capturing Application)
  A user who access the middleware to retrieve RFID events. This is usually a business application.
- Interrogator (RFID Reader)
  The interrogators are the devices with access to RF tag for reading and writing data.
- Subscription
  A subscription consists of patterns of IDs to include or exclude in the notification report. It also has NotificationURI as the destination for generated reports.
- Filtering & Collection (F&C)
  An intermediate software system that operates between subscribers and interrogators. This is the middleware that this thesis focuses on.
- Notification
  A report generated in F&C upon the observance of subscribed events from interrogators. The report is then delivered to the corresponding subscriber.
- Reader Cycle
  A reader cycle is the smallest unit of interaction with an RFID reader.
- Event Cycle
  An event cycle is an interval of time over which F&C carries out interactions with one or more interrogators.

A typical flow of RFID events is described in FIGURE 2.9. Firstly the event capturing applications (subscribers) subscribe ID pattern filters in the middleware with a NotificationURI, and a report is notified to the URI upon a matching RFID event arrival.

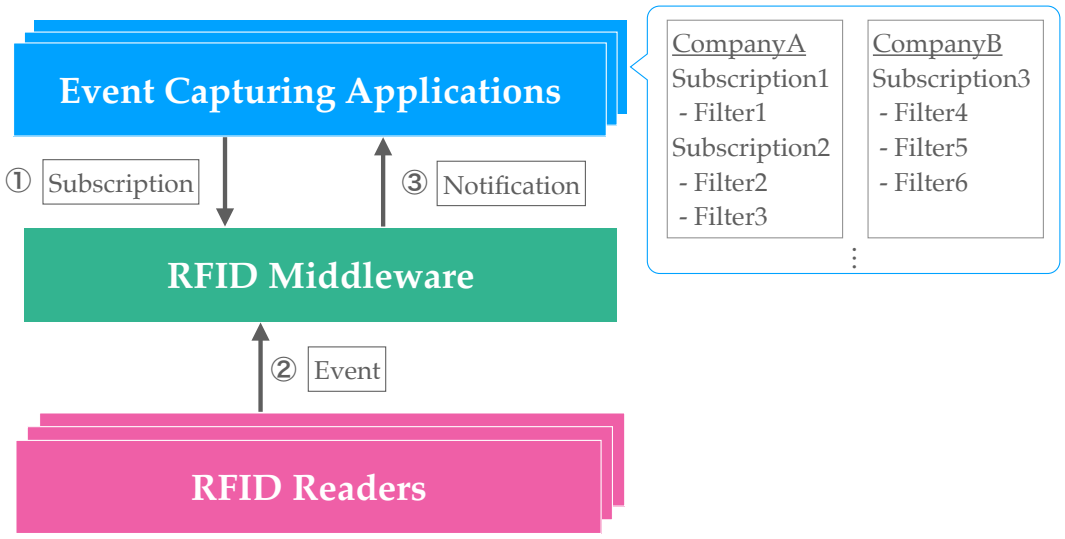


FIGURE 2.9: Users (e.g., companies) subscribe interested ID patterns from the middleware and receive Notification reports.

2.3.2 Application Level Events (ALE)

Application Level Events (ALE) is an interface for subscribers to register their interested ID patterns and NotificationURIs. The importance of this interface is for the business applications to have an access to the lower layer (interrogators) without configuring the hardware and make use of logical indicators to select the data source. ALE uses a WSDL (W3C, 2007) to define, configure, ad request reports from the middleware through the API expressed as XSD (W3C, 2012). The subscribers first registers an logical reader with LRSpec (FIGURE 2.10) and subscribes for ID patterns with ECSpec (FIGURE 2.11).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:LRSpec xmlns:ns2="urn:epcglobal:ale:wsdl:1"
 ▯→  xmlns:ns3="urn:epcglobal:ale:xsd:1">
  <isComposite>false</isComposite>
  <readers/>
    <properties>
      <property>
        <name>ReaderType</name>
        <value>github.com/iomz/go-llrp/llrp-adapter</value>
      </property>
      <property>
        <name>Description</name>
        <value>TerminalGateA</value>
      </property>
      <property>
        <name>PhysicalReaderName</name>
        <value>TerminalGateA-12345 </value>
      </property>
      <property>
        <name>ReadTimeInterval</name>
        <value>10000</value>
      </property>
      <property>
        <name>ip</name>
        <value>192.0.2.1</value>
      </property>
      <property>
        <name>port</name>
        <value>5084</value>
      </property>
  </properties>
</ns3:LRSpec>
```

FIGURE 2.10: LRSpec to register a logical reader in the middleware. This request registers the LLRP reader operating at 192.0.2.1.

The report notified to the NotificationURI is also in an XSD format called ECReports as shown in FIGURE 2.12.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ECSpec xmlns:ns2="urn:epcglobal:ale:xsd:1">
  <logicalReaders>
    <logicalReader>TerminalGateA</logicalReader>
    <logicalReader>TerminalGateB</logicalReader>
  </logicalReaders>
  <boundarySpec>
    <repeatPeriod unit="MS">10000</repeatPeriod>
    <duration unit="MS">9500</duration>
    <stableSetInterval unit="MS">0</stableSetInterval>
  </boundarySpec>
  <reportSpecs>
    <reportSpec>
      <reportSet set="CURRENT" />
      <output includeTag="true" includeCount="true"/>
      <filterSpec>
        <includePatterns>
          <includePattern>
            urn:epc:pat:sgtin-96:3.458960468.*.*
          </includePattern>
        </includePatterns>
        <excludePatterns>
        </excludePatterns>
      </filterSpec>
    </reportSpec>
  </reportSpecs>
</ns2:ECSpec>
```

FIGURE 2.11: ECSpec for subscribing ID filtering patterns with EC-FilterSpec. This request includes the pattern of SGTIN-96 POS ID of Company Prefix “458960468”.

2.3.3 Low Level Reader Protocol (LLRP)

Low Level Reader Protocol (LLRP) is a protocol ratified by GS1, 2010 and also extended in ISO/IEC, 2012. LLRP is commonly supported by commercial RFID readers to manage and configure them to provide access to RFID air protocol commands and their respective command parameters in binary transfer syntax over TCP/IP. It is also designed to allow the LLRP client to make direct read/write access to all date on an RF tag, e.g., lock tag memory banks, kill tags, and access raw-binary RF tag data on a tag for both reading and writing.

LLRP always establishes an end-to-end connection between a reader and a client, and either of them can initiate the communication. The client then commands the reader to configure how the reader shall operate for RF tag inventory with parameters; specific memory bank o□sets to be read, inventory intervals, antenna power, etc. The reader then starts the configured inventory operation.

As we take a look at LLRP packets, observed RF tags are reported in a message RO_ACCESS_REPORT (FIGURE 2.14) inside the parameter TagReportData (FIGURE 2.15) as a parameter EPCData (FIGURE 2.17) or EPC-96 (FIGURE 2.17) depending on the type of the RF tag.

```
<?xml version="1.0"?>
<ECReports xmlns="urn:epcglobal:ale:xsd:1" schemaVersion="1.1"
  creationDate="2018-01-11T05:09:06.012-09:00" specName="ECSpec1"
  date="2018-01-11T05:09:06.012-09:00" ALEID="ALEID_1"
  totalMilliseconds="9500" terminationCondition="DURATION">
  <reports>
    <report reportName="ReportName1">
      <group>
        <groupList>
          <member>
            <tag>
              urn:epc:tag:sgtin-96:3.458960468.0022.86
            </tag>
          </member>
        </groupList>
        <groupCount>
           <count>1</count>
        </groupCount>
      </group>
    </report>
  </reports>
  <ECSpec creationDate="2018-01-11T05:08:54.222-09:00"
     schemaVersion="1.1">
     <logicalReaders>
        <logicalReader>TerminalGateA</logicalReader>
     </logicalReaders>
     <boundarySpec>
        <repeatPeriod unit="MS">10000</repeatPeriod>
        <duration unit="MS">9500</duration>
     </boundarySpec>
     <reportSpecs>
        <reportSpec reportName="ReportName1">
           <reportSet set="CURRENT" />
           <output includeTag="true" includeCount="true" />
        </reportSpec>
     </reportSpecs>
  </ECSpec>
</ECReports>
```

FIGURE 2.12: ECReports to be notified from the middleware.

2.3.4 Filtering Engine

The primary role of the middleware is to filter the IDs extracted from RFID events by the subscribed patterns. I name the module Filtering Engine to be responsible for the ID filtering process. A filtering engine is a search engine for an ID to find one or multiple patterns registered in the middleware by the subscribers. Naturally, it is composed of search algorithms and the structure of the filters is implemented as various data structure by di□erent search algorithms; that could be a list or a tree. As long as a filtering engine provides the same interface for the LLRP to hand over IDs and is able to report the matched IDs to the ALE module, the structure doesn't

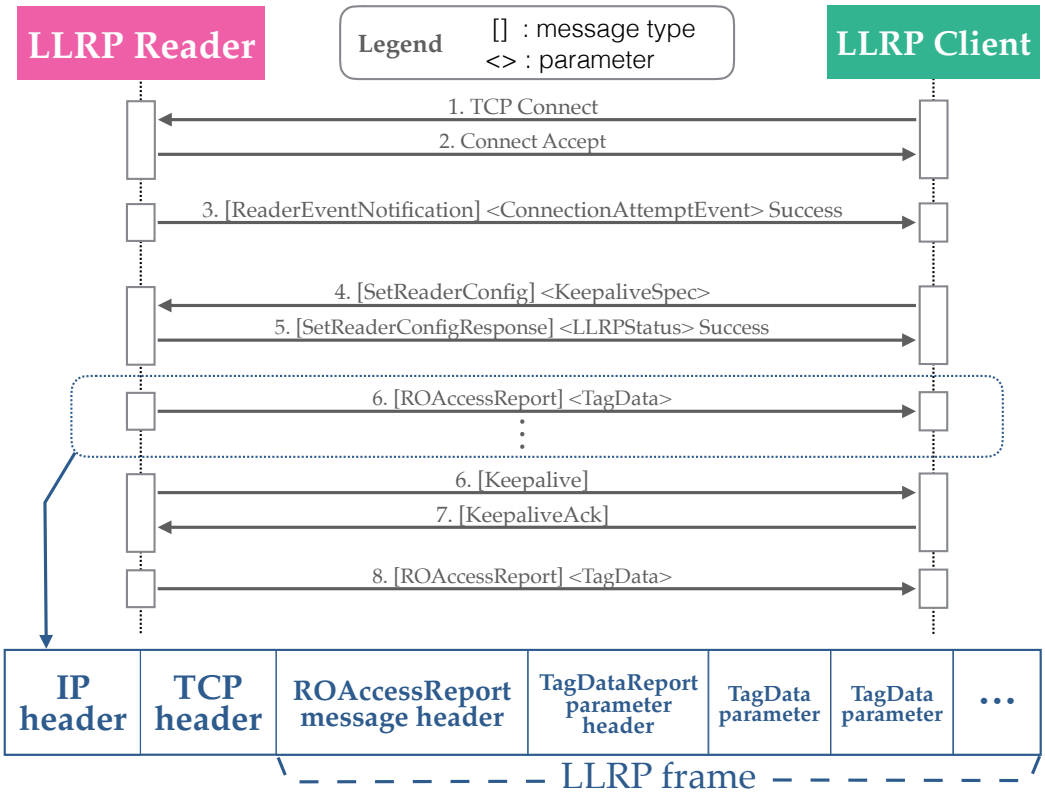


FIGURE 2.13: Sequence diagram for a client initiated LLRP communication and the LLRP frame format for an ROAccessReport message containing data from RF tags.

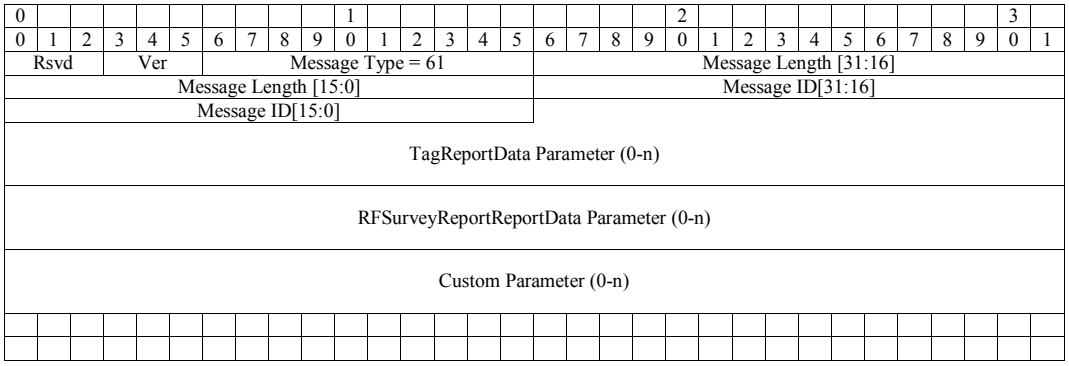


FIGURE 2.14: Binary encoding of RO_ACCESS_REPORT message from GS1, 2010. This message is issued by the Reader to the Client, and it contains the results of the RO and Access operations based on a configuration from the ROReportSpec and AccessReportSpec parameters.

really matter (2.18).

Further detail of a filtering engine is later discussed in Chapter 5.

2.3.5 Related Commercial RFID Middleware Products

At the time of writing, there are only a few commercial RFID middleware on the market. RECO-Bridge IDR-1 V2 from Ricoh Company Ltd., 2015, is the software

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Reserved						Type = 240										Length															
EPCDataParameter [See notes below]																															
ROSpecID Parameter (0-1)																															
SpecIndex Parameter (0-1)																															
InventoryParameterSpecID Parameter (0-1)																															
AntennaID Parameter (0-1)																															
PeakRSSI Parameter (0-1)																															
ChannelIndex Parameter (0-1)																															
FirstSeenTimestampUTC Parameter (0-1)																															
FirstSeenTimestampUptime Parameter (0-1)																															
LastSeenTimestampUTC Parameter (0-1)																															
LastSeenTimestampUptime Parameter (0-1)																															
TagSeenCount Parameter (0-1)																															
AirProtocolTagDataParameter (0-n)[See Notes below]																															
AccessSpecID Parameter (0-1)																															
OpSpecResultParameter (0-n) [See notes below]																															
Custom Parameter (0-n)																															

FIGURE 2.15: Binary encoding of TagReportData parameter from GS1, 2010. This parameter is generated per tag per accumulation scope with the mandatory parameter EPCData.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Reserved						Type = 241										Length															
EPCLengthBits																															
EPC																															

FIGURE 2.16: Binary encoding of EPCData parameter from GS1, 2010.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
1	Type=13							EPC[95:72]																							
EPC[71:40]																															
EPC[39:8]																															
EPC[7:0]																															

FIGURE 2.17: Binary encoding of EPC-96 parameter from GS1, 2010. A variance of EPCData parameter. For 96-bit EPC, this parameter is used instead of EPCData parameter in TagReportData.

solution operated in Microsoft Windows Server 2012. The middleware is written in Java, and is capable of housing 32 RFID readers connecting to 128 antennas, with LLRP and ALE supported. The throughput of the input and output is not disclosed. The price range is from 980,000 JPY.

WebOTX RFID Manager Information Service is also a middleware solution from NEC Corporation, 2015. This middleware is written in Java and both Windows and Linux system are supported. It is designed to intercommunicate with other NEC's WebOTX family products and provides the LLRP and ALE interface as well. No performance threshold is mentioned, but by looking at the required hardware specs the a□ordable tra□c or the number of filters is not in the order of the requirements. The price range is from 100,000 JPY.

EasyTap is another RFID middleware from Quake Global, Inc., 2017. It seems capable of visualizing the tra□c of RFID events with the support of LLRP, but no advanced document was found.

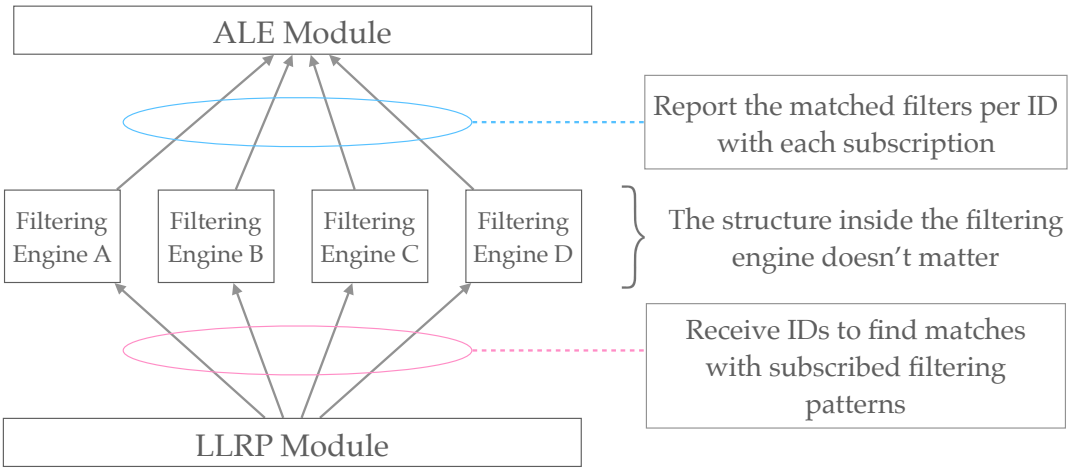


FIGURE 2.18: Structure of various filtering engines with a common interface.

Transcends, LLC., 2016 provides a middleware as a part of RIFIDI Edge Server. RIFIDI used be open to public a couple of years ago, and they serve as an RFID platform integrator. It is a set of software solution, but I couldn't a□ord to try the actual deployment.

There are a couple of other RFID middleware solutions: SAP AG., 2007 and Oracle, 2007, but due to the lack of the up-to-date information, I was not able to investigate them further.

RFID readers on the other hand, there are many vendors developing for sale (Alien Technology, LLC., 2018; FUJITSU FRONTECH LIMITED, 2018; Honeywell International Inc., 2018; Impinj, Inc., 2018; Mitsubishi Electric Corporation, 2018; OMRON Corporation, 2018; TAKAYA CORPORATION, 2018; TOSHIBA TEC CORPORATION, 2018; Tyco Sensormatic, 2018; ZIH Corp, 2018), and most of the products are supporting the LLRP interface for the management of their reader devices.

Chapter 3

Preliminary Experiment

In this chapter, I discuss preliminary experiments conducted with the existing open source RFID middleware Fosstrak F&C. To perform an evaluation on Fosstrak implementation, I also implemented an RFID reader emulator golemu to establish a simulated environment of RFID event streams. After the observation of the experiments, I establish the criteria for a new robust and high-performance RFID middleware discussed in the rest of this thesis.

3.1 Performance Measurement by RFID Reader Emulation

When deploying RFID-based system, testing environment accommodation is significantly challenging. The hardware takes diverse parameters like the number of antennas, readers, the duration of Event Cycle, the bandwidth of interrogation, the output power from antennas, etc. In fact, this explains why RFID middleware vendors are not disclosing performance specifications in their catalogues (see Section 2.3.5), as it is quite sophisticated to define a testing environment.

RIFIDI (Huebner, Facchi, and Janicke, 2012) is an open source RFID/Sensor Middleware Platform sponsored by Transcends. The toolkit of RIFIDI called Tag Streamer o□ers test suite of physical RFID reader simulation in the virtual framework powered by RIFIDI emulator. The test suite randomly generates tags with IDs as desired and then the virtual inventory is performed along with designated scenarios. However, the test suite is not capable of holding patterns of RFID events with PC bits and other auxiliary data required for multi-code discrimination according to the community documentation[1]. RIFIDI Tag Streamer is only intended for EPC tags so it also cannot generate RFID events of variable length ID such as ISO's. Therefore RIFIDI is not suitable for testing multi-code compatible information systems.

RFID Middleware Evaluation Toolkit by Park, Ryu, and Hong, 2009 is another candidate, but regrettably it is currently not publicly available so it cannot be usable either.

3.1.1 golemu – An RFID reader emulator

In order for the multi-code information system to validate the filtering rule and system robustness, a certain amount of multi-code RFID event simulation is necessary. I implemented an RFID emulator golemu to fulfill the following requirements as a testing tool to help the robust RFID middleware development and testings.

[1] http://wiki.rifidi.net/

1. Conformable to LLRP version 1.1
2. Emulate multi-code RF tag read events encoding GS1 Tag Data Standard, ISO/IEC 15459, 15961, ISO 1736x, and other proprietary IDs
3. Each RFID read event is able to carry C1G2 Air Protocol specific parameters (PC bits) and specified user memory data
4. The tag population per an emulator can be dynamically modified while the LLRP connection is active to emulate the actual situation of reading events

The emulator is purely written in Go and publicly available on GitHub[2]. Even though there is a widely known LLRP Library called LLRP toolkit[3], it is not actively maintained with open contributions. Hence, I also built a simple LLRP library for Go[4] which only supports essential portions of the whole specification required to establish an LLRP connection with a client and generate ROAccessReport of virtual RF tags. The compiled emulator is a single-binary execution file and it can run on any kinds of modern OS that supports Go runtime environment (FreeBSD, Linux, macOS, Windows 32-bit/64-bit) that allows the emulator to be deployed on site with ease and portability.

The flow of multi-code RF tag read events from the emulator is designated to the RFID middleware. Various kinds of tags are reported to the middleware for filtering and translation into URI format, then they are reported to subscribed applications. The virtual tag population control is accessible through the web interface or the API. FIGURE 3.1 shows the internal modules in golemu. When simulating virtual RFID events, the API is used to produce specific scenarios (FIGURE 3.2). golemu is portable as it doesn't require any dependencies other than Go runtime environment and some libraries. An RFID inventory simulation with golemu has been demonstrated at various venue (FIGURE 3.3).

With this emulator I have evaluated the Fosstrak F&C middleware to see the upper performance limit and discovered the bottlenecks in the components. Since Fosstrak F&C middleware is written in Java, I used Java Mission Control tool chain[5] (FIGURE 3.4).

From the experiment, the Fosstrak implementation can a□ord up to 10,000 IDs for only three filters to capture all the EPC, ISO, and the rest of non-standard IDs. As the ID tra□c getting closer to 1,000/sec, the notification reports start to delay and become unstable. Sometimes it drops a part of or entire RFID read events in the event cycle. There are many factors causing this fragility, but I frame out the implementation-oriented matters and focused on the general design structure of the F&C. FIGURE 3.4 shows the top two sampled package call counts during one minute of continuous filtering the given number of IDs at 10-second event cycle. We can observe the sub-peaks for the both packages before hitting 10,000 IDs and the number of calls dropped afterward, and gradually increase again.

The guideline for the use of golemu is attached as appendix C.

[2] https://github.com/iomz/golemu
[3] https://sourceforge.net/projects/llrp-toolkit/
[4] https://github.com/iomz/go-llrp
[5] http://www.oracle.com/technetwork/java/javaseproducts/mission-control/java-mission-control-1998576.html

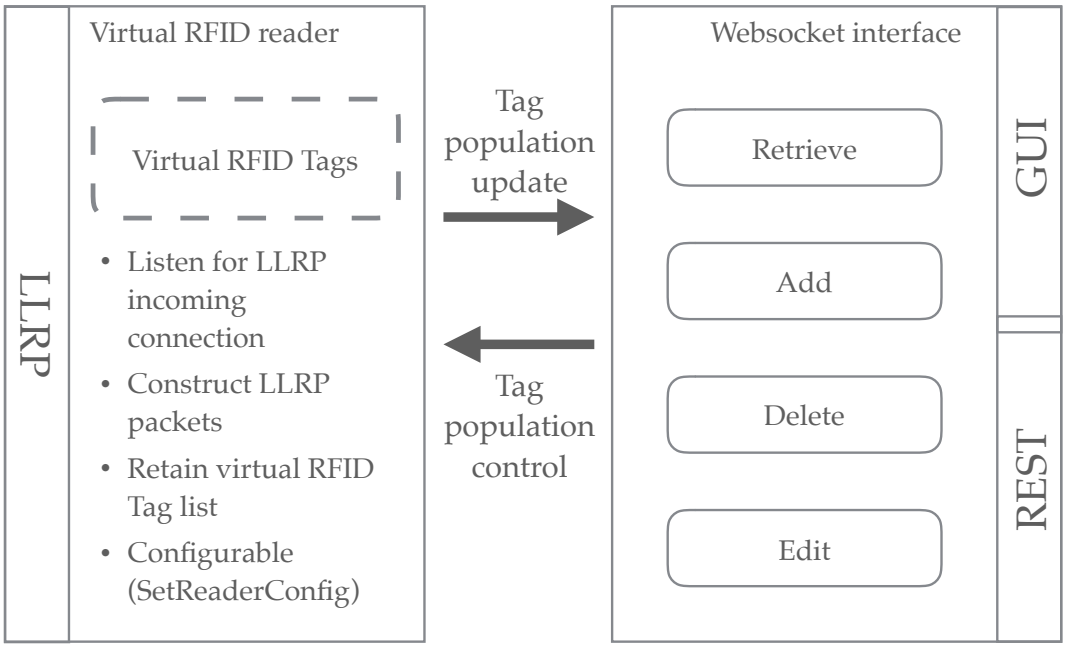


FIGURE 3.1: Overview of golemu's software design and interfaces. golemu holds virtual population of RF tags and can be easily interacted through both the web interface via Websocket or the REST API.

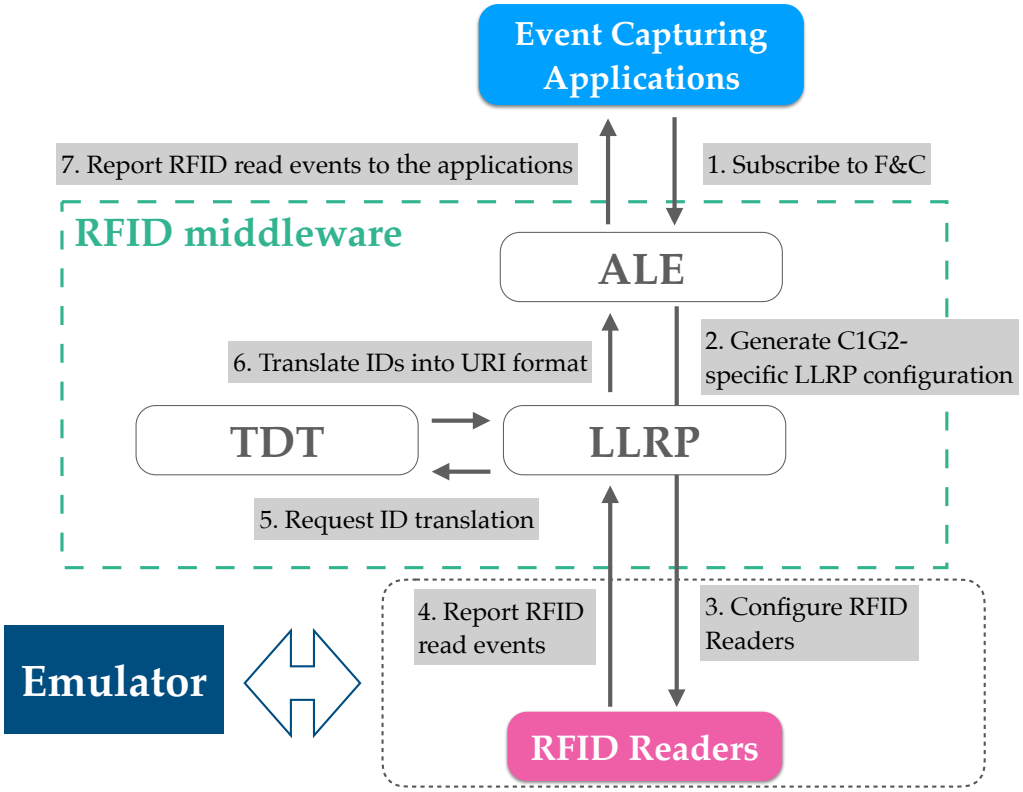


FIGURE 3.2: Overview of an RFID middleware mechanism to discriminate RF tag standards and the LLRP emulator in place of physical RFID readers. The emulator reports virtual multi-code RF tag read events externally controlled.

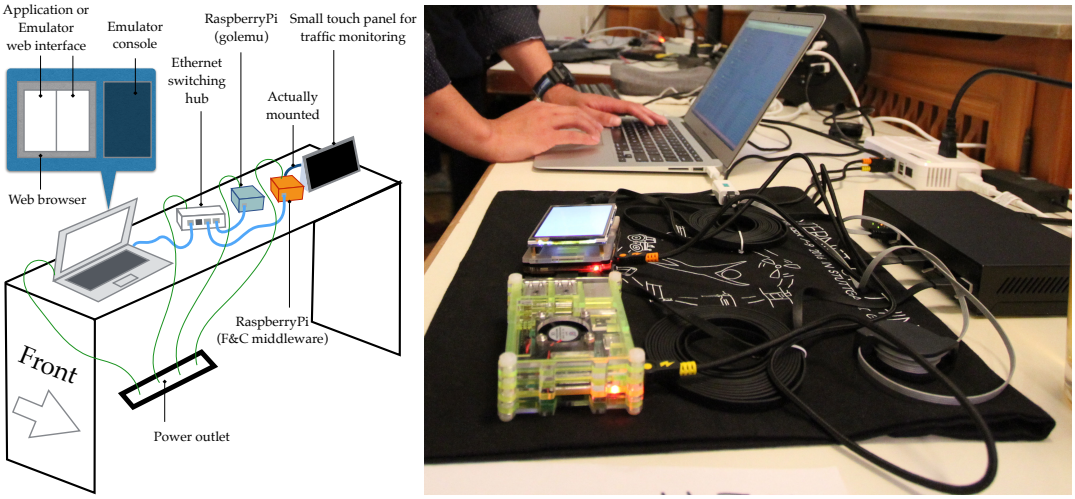


FIGURE 3.3: golemu deployed to the Raspberry Pi with a mini display. It is connected to the F&C middleware also deployed to another Raspberry Pi in the yellow casing via the network switch and simulating RFID inventory.

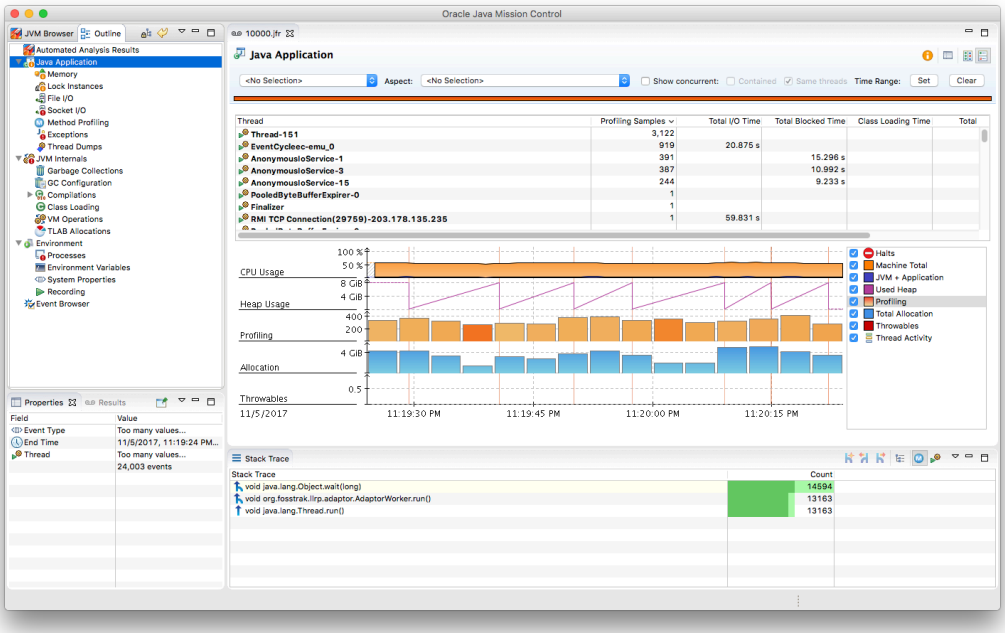


FIGURE 3.4: Screenshot of Java Mission Control showing the profiled summary for a 10,000 ID scenario captured by Java Flight Recorder tool.

3.2 Observation Analysis

In this section, I discuss four problems identified from and through the experiment.

3.2.1 Notification Delay

The first and obvious “symptom” when nearing the upper performance limit was the delay in the notification reports. In Fosstrak, the notification timing is synchronized

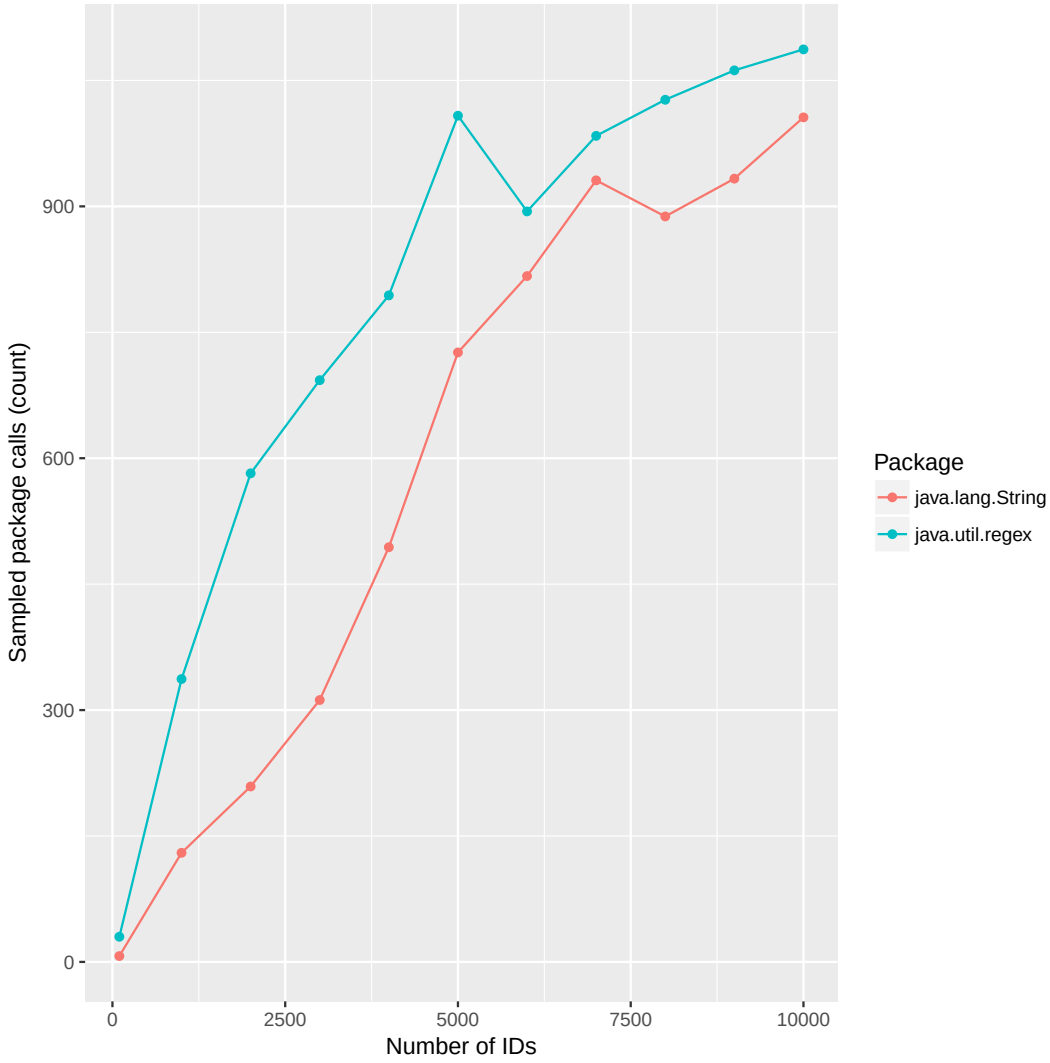


FIGURE 3.5: The top two most sampled method calls per package in Fosstrak implementation by the number of IDs.

with the end of each configured event cycle. On the contrary, the event cycle doesn't repeat asynchronously with the ID filtering process and paused until the filtering job finishes. Considering that Fosstrak is the prototype for presenting standard-conformable F&C middleware back then, it was quite natural to think constant 1,000 IDs/sec was a overkill, since even now no enterprise-level RFID system is operating with that high tra□c. However, I aim for the robustness capable of anticipated high tra□c in the future, and if this vulnerability is left the realtimeness of the notification report is anyway lost in-between the event cycles with heavy ID tra□c. Once the delay occurs whilst the amount of tra□c maintains, the delay propagates to the next cycles and a□ects all the following notification (FIGURE 3.6).

To solve this issue, the middleware needs to (a) *cut o* unfinished filtering process and drop the IDs unfiltered to fit in the event cycle, (b) carry over the unfiltered IDs to the next event cycle, or (c) implement the mechanism to o□oad incapacitated filtering process and force them fit within the event cycle. The approach (a) let the subscribers lose events which possibly cannot be observed again so is not an option. (b) keeps the under processed events by cascading to the following event cycles, but

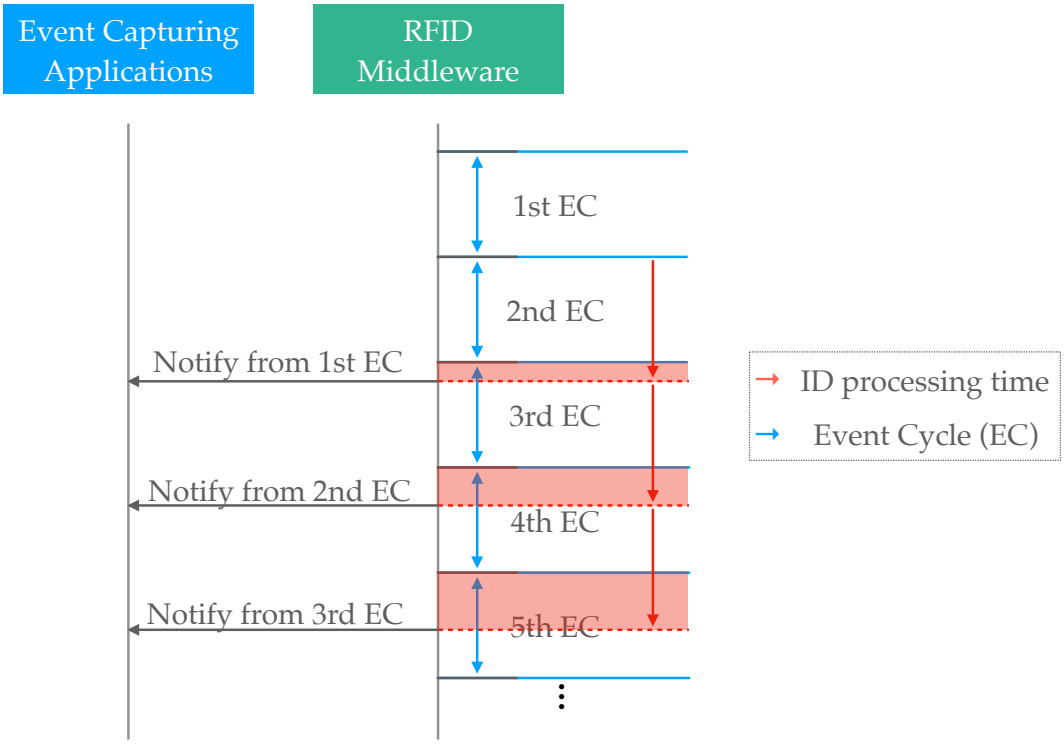


FIGURE 3.6: The delay of ID processing propagates toward the future event cycles.

it also lose the realtimeness. Consequently, (c) is the only choice for the objective.

3.2.2 Linearly Proportional Iteration and Variable Conversions

Intuitively, as the number of IDs or filters grow, the iteration of internal comparison increases. In the implementation, the filtering engine is a list of appended strings. The ID in ROAccessReport in the LLRP data-gram is in binary format and every time an ID is extracted the packet, it is converted to its string representation, translated into a URI format, and then the comparison is finally made. To pursue the high-performance comparison, this chain of conversion is unnecessary if the ID doesn't match with any filter as it is discarded. Furthermore, from FIGURE 3.5 we know the string and regex are the two most called packages, which are obviously handling this conversion. The resolution for this problem is to reduce the number of unfinished conversion of the IDs by first performing byte-wise comparison with subscribed filters, which are also converted in binary format beforehand, and only translate the matched IDs. As I mentioned in Chapter 2, EPC and ISO have di□erent binary encoding schemes, but in a binary format of IDs can be distinctive to each other and we don't even think about the translation at least during the matching process. This approach improves the overall throughput of the comparison part, and in addition, we can also think about how the filters should be structured otherwise than the list.

3.2.3 Redundant Subscriptions

Another consideration that needs to be taken into account is the redundancy in the subscribed filters. In FIGURE 3.7, there are four filters. Each has the common part with other filters; e.g., the blue part is common among all the filters. When these

filters are subscribed to the middleware, the question is whether we want to check those parts every time trying to match with each filter. The redundant pattern in those filters is in other words a room for improvement, for instance, Filter1 and Filter2 are almost identical except for the one bit, IDs matched with Filter1 is a subset of Filter3, what if we make another filter with the blue colored pattern to shake down uninterested IDs, and so forth.

FIGURE 3.7: Redundant filters in subscriptions.

Although it is challenging to come up with a solid algorithm to generate the most optimized way of removing the redundancy with more filters, it is worth analyze the possible procedure of doing it. When making a prefix-based filtering engine, a "worm-eaten" pattern like the blue part doesn't fit with the search algorithm, but we can make use of the subsets to reducing the bifurcation in the tree structure.

3.2.4 Biased ID population

Lastly the actual population of IDs stream into the middleware is an important factor. In reality, the number of IDs matched with subscribed filters is not the same, in fact there could be huge gaps. For instance, in the convenience store, hot-selling product shall be read more frequently than minor items as it is through the POS system, and concomitantly the store might have proportional amount of stocks to the trend. In a logistics center example, some supplier handles large but small amount of products while others have small but large amount product line. For the skewed ratio of matching filters, the filtering engine should systematically optimize for the frequent matchers. By its very nature of inventory management system, possible RFID-based information systems do have the estimation of the ID population; e.g., convenience stores have the number of each items in POS system and delivery operators registers the item when they put the package in transit line. Given that kind of information, the middleware can be enhanced further to adopt environmental characteristics to absorb the ine□ciency from the biased ID tra□c.

3.3 Criteria for Robustness and High Performance

From the observations in the previous section, I define the following criteria and scenarios. The rest of the thesis is based on the following scenarios to integrate Entropy Filtering in the RFID middleware.

- Robustness and Portability
  The robustness of the middleware should be guaranteed against unexpectedly heavy traffic of IDs. The deployment environment for the middleware differs in use cases, so as to keep the deployment procedure simple, the middleware shouldn't require an enterprise level hardware.
- High throughput filtering
  The scalable performance is essential for this study. The filtering process should take as minimum as possible to gain the throughput of ID/sec.

Scenario 1

In international freight container warehouses where cargo units are handled in ISO IDs while products inside are managed in GS1 IDs. Warehouse administrator wants to manage those multi-code RF tags in an unified information system with an RFID middleware. The subscribing event capturing applications are interested in different IDs and some of them need to write in the user memory in RF tags to manage freight containers. The warehouse has 100 RFID readers installed and they are expected to read 100 IDs each at maximum. As the reader also needs to perform write operation in some of the RF tags, the acceptable duration to identify the matched filter per ID is below 1us. The subscription update interval is days to weeks, and the characteristics of subscribed filters are mostly Company Prefix level, but the number of IDs read per day is the order of 1,000,000.

Scenario 2

A convenience store has an RFID auto-checkout station and shoplifting prevention RFID reader gate. Every product in the store has an RF tag attached to its exterior and used for both checkout and theft prevention. Once customers bring the items to the station, the check-out system reads the items to be purchased, and delete them from its subscriptions which are in the RFID middleware connected with the POS system. After the purchase, the filtering engine is rebuilt by the time customer leaves the store and the theft prevention gate will not detect the purchased items as stolen. Average number of items in a convenience store is 2,800 with 100 stocks each, yielding approximately 300,000 unique items and so is the number of filters in the filtering engine. The worst case is just after the timing of stock, 3 customers buy 100 items each, and it takes one minute for them to leave the store. Hence, the middleware should handle 300 ID matches in the filtering engine contains 300,000 filters and remove matched filters under 5 seconds.

Scenario 3
Two clothing stores of different marketing bodies are tenanted inside a shopping mall next to each other. Each store has an RFID based system for their inventory management. The two stores both have chosen GS1 IDs for their items and every IDs are conformable to the standard specifications. However, the items inside the stores can be read by RFID readers of both stores and it is required to filter IDs under specific rules in order to keep the management system clean. The RF tags are sewed inside the fabrics and both stores have 100,000-300,000 items in-store. Since they are operated separately, they cannot know the other's inventory information so should expect unknown amount of IDs to filter out.

Chapter 4

Entropy Filtering

Shannon, 1948 proposed a mathematical concept called *information entropy* inspired from thermodynamics. The entropy H is defined when $p_1, p_2, \ldots, p_n$ are the probabilities to su□ce $p_1 + p_2 + \cdots + p_n = 1$:

$$H(p_1, p_2, \ldots, p_n) = \sum_{k=1}^{n} p_k \lg \frac{1}{p_k}$$

In the context of RFID middleware, p_n is the probability for an arbitrary ID to match a subscribed filter $pattern_n$. Intuitively, as the number of subscribed filters increases or more uneven the probabilities distribute, the entropy H becomes high and it takes time for the middleware to execute the filtering process. To the best of author's knowledge, the entropy can be a□ected not only by the number of filters, but also by other parameters like the rate of incoming IDs, the average length of all the filters, and the similarity, redundancy, or containment relationship of the filters.

The ultimate goal of Entropy Filtering is to construct a filtering engine, which can maintain the entropy as low as possible at all times. An RFID middleware is a real-time system, so as the environmental factors change and the state of the middleware also receives the consequences. Similarly but in a di□erent context, Burgard, Fox, and Thrun, 1997 used this concept to detect a mobile robot's localization by utilizing sensor data to generate the entropy for the robot's current position, feedback the actuators to support the localization estimation, and repeat the procedure by minimizing the entropy of the robot location. Such feedback cycles enable the middleware to improve the filtering performance by dynamically adjusting itself to the entropy.

The rest of this chapter explains the Entropy Filtering method in the filtering engine of an RFID middleware.

4.1 Parameter Assessment and Proposition

The entropy is a measure of uncertainty for how likely the filters in an RFID middleware to match IDs, and the diversity index has macroscopic properties that may be usable to boost the filtering engine performance. In other words, Entropy Filtering uses the entropy as a diversity index to evaluate the filtering engine in accordance with the criteria and re-engineers the engine whenever necessary. As mentioned earlier in this chapter, the following parameters can be regarded as macroscopic properties for the entropy.

- Number of filters (subscriptions)
  The number of filters a□ects the time to find the subscribed patterns to match IDs. It is also an important factor for the time for rebuilding the engine depending on the algorithm of the filtering engine. In particular, the number of subscribed filters is the most influential factor for the time both to build a filtering engine and to filter an ID to find matches.
- The length of filters
  When the average length of subscribed filters is short, the part of ID to be compared remains short as well.
- Containment relationship of filters
  The containment relationship between the filters impinges on the redundant execution of filtering process. This is directly the cause of the redundant subscription problem mentioned in Clause 3.2.3.
- Subscription frequency
  The frequency of the subscriptions is the most distinctive parameters from case to case. A use case with frequent subscriptions anticipated requires the filtering engine to be capable of recurrent rebuilding.

As a methodology, I propose a conceptual flow of the dynamic feedback cycle as shown in FIGURE 4.1. There are three players in the circulation: Processor executes the job, Collector monitors the status of operation, or the change in external systems, and Advisor receives feedback from Collector and manage Processor.

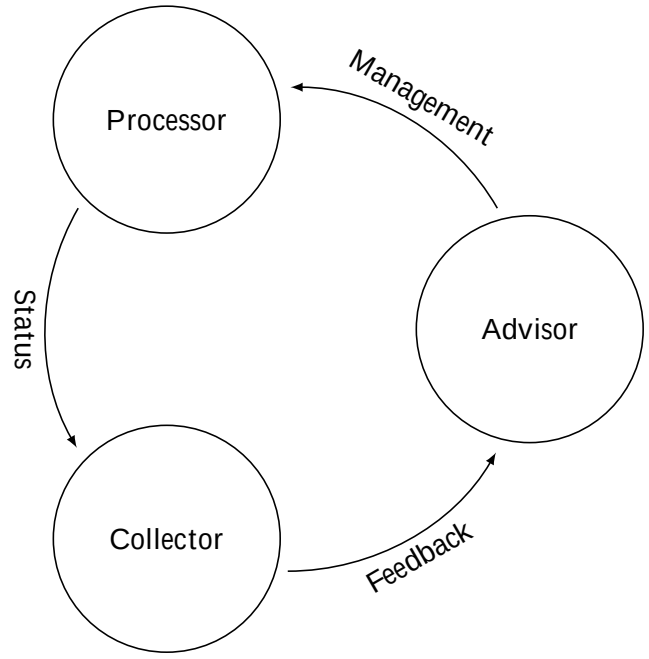


FIGURE 4.1: A concept of dynamic feedback cycle for Entropy Filtering.

4.2 Implementation of Entropy Filtering

Entropy Filtering collects information about each of several different environmental factors and assess their individual contributions to the total entropy to rebuild the filtering engine. The feedback cycle within the middleware functions as the mechanism to improve the performance of the filtering engine in timely manner. Recapturing the structure of filtering engine, it can be any data structure as long as it provides the common interface with the ALE and LLRP modules. The filtering engine is responsible for filtering the IDs streamed from RFID readers and reporting the captured events to each subscriber.

FIGURE 4.2 shows the implementation of Entropy Filtering in an RFID middleware by reflecting the feedback cycle on the message flows. The sequential diagram shown in FIGURE 4.3 explains the timing of rebuilding and updating the algorithm in the filtering engine based on the external factors. In the diagram, three environmental factors are collected by **Monitor Node**: the traffic rate of LLRP packets from the LLRP module, the resulting filtered IDs and locality of the subscribed filters from ALE module, and scheduled (anticipated) population of RFID tags that will be processed by the RFID readers connected to the middleware. The accumulated information is then supplied to **Master Node** which then decide whether or not the filtering engines need to be rebuilt.

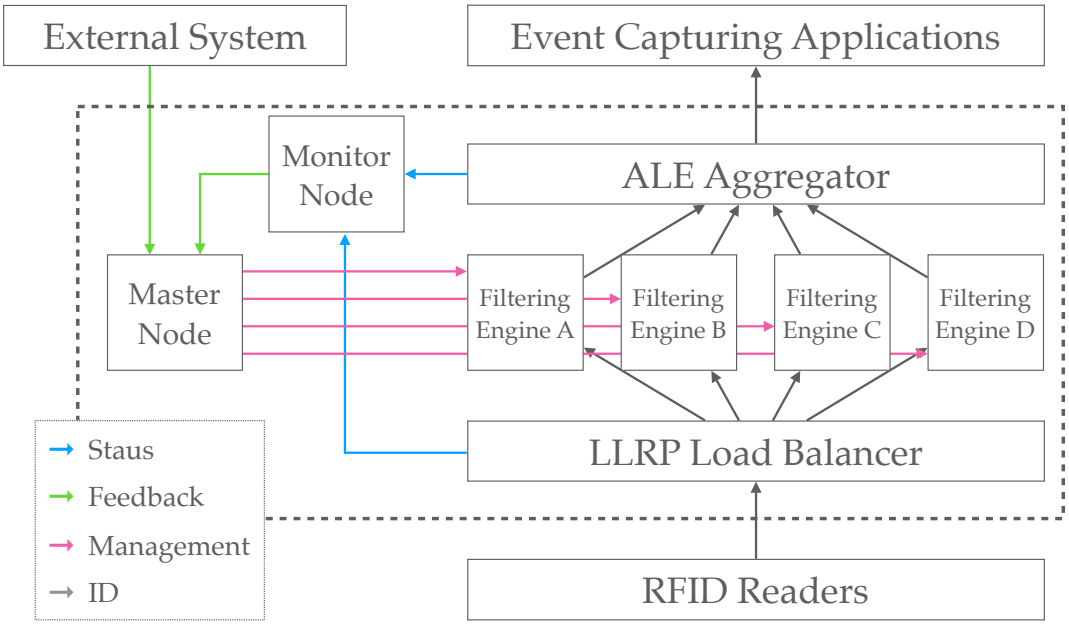


FIGURE 4.2: Overview of Entropy Filtering implementation for an RFID middleware.

Recalling from Chapter 2, the IDs have the following characteristics:

a) **Binary encoded**
IDs are stored in the specific memory bank of RF tags and they are binary encoded in order for RFID readers to read effectively.

b) **Variable length**
IDs have variable length; each coding scheme has predefined length for the ID.

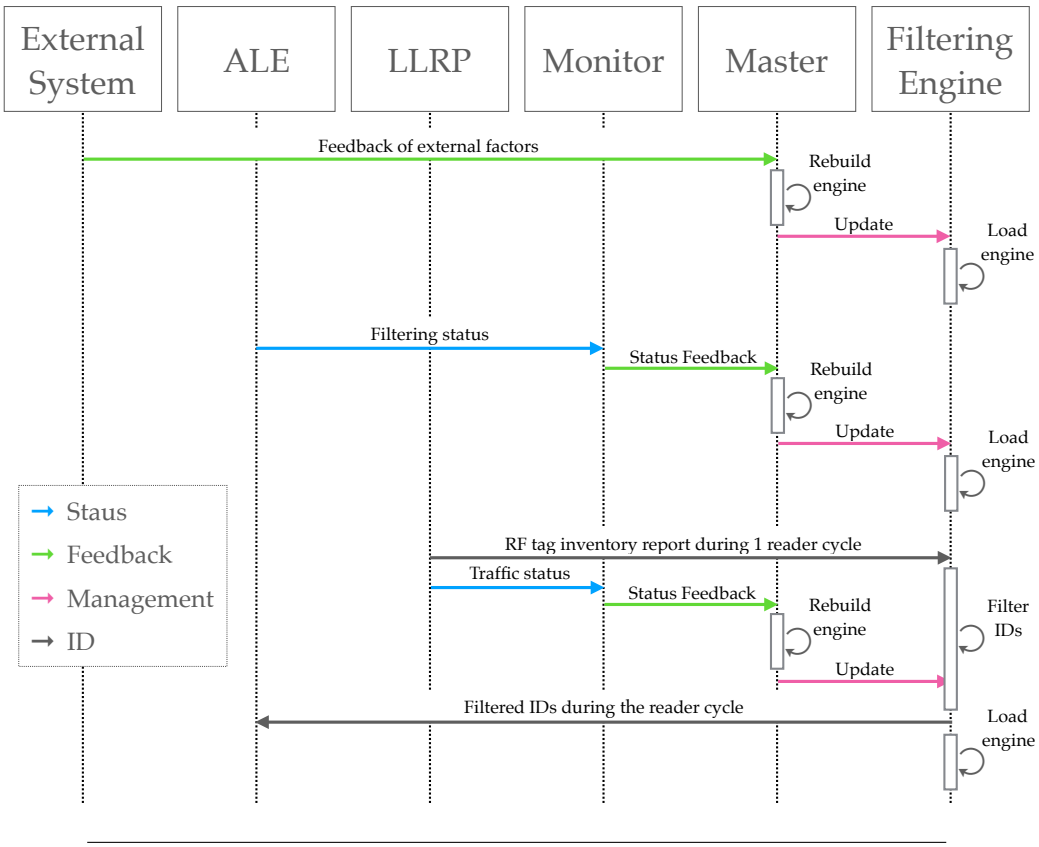


FIGURE 4.3: Sequential diagram of Entropy Filtering in an RFID middleware.

ISO coding schemes even allow an ID to be any size as long as it doesn't exceed the limit.

c) Hierarchical
Every standard has the hierarchical structure. The filters are using this characteristics to e□ciently identify a group of IDs that are of interest.

d) Not always valid
We should assume there are invalid RF tags in the practical situation.

The potency of Entropy Filtering is demonstrated by evaluating filtering engine with algorithm that is optimized for specific use cases defined as the simulation scenarios in section 3.3.

4.2.1 gosstrak-fc – A Filtering & Collection RFID middleware

In order to evaluate the filtering engine, I have launched a project gosstrak-fc[1] for an RFID middleware written in Go. At the time of writing, it has only the filtering engine module and lacks the LLRP and ALE interface, but it is intended and designed to be extended as a fully standard-conformable Filtering & Collection middleware. The synopsis and the implementation of the algorithms in the next section for gosstrak-fc is explained in appendix D.

[1] https://iomz.github.io/gosstrak-fc/

4.3 Filtering Algorithms

In this sections, I introduce four algorithms to be part of the filtering engine for Entropy Filtering. All the algorithms receives an ID as input and yields a list of matching patterns as output. The latter three algorithms include recursive procedures, and so are presented in pseudo code format with functions as well.

4.3.1 Linear Search (List)

Linear Search algorithm finds matched patterns simply by the iteration of a list of all the subscribed filters (Algorithm 4.3.1). Among the algorithms in this chapter, Linear Search is the only algorithm that can accept non-prefix pattern filters as discussed in Chapter 2. Since there is no bifurcation or branching to eliminate unnecessary look up, it takes $O(n)$ to find matches for an ID where n is the number of patterns subscribed. From a different point of view, as this algorithm doesn't require any sophisticated data structure for the filters, the time to take for building a list-based filtering engine is almost negligible in comparison to the other engines.

Algorithm 4.3.1. Linear Search

```
Filters         : A dictionary of value notificationURI by key pattern
id              : An ID to lookup
notificationURI : A string contains subscribed notification URI
pattern         : A byte sequence of ID pattern
1 for (pattern, notificationURI) ∈ Filters do
2     if id has pattern then
3         Notify id to notificationURI;
```

4.3.2 Patricia Trie

Patricia Trie is first introduced by Morrison, 1968 and also known as Radix-tree. The algorithm is used in a wide range of area: IP address look-up (Yilmaz et al., 2000; Ruiz-Sánchez, Biersack, and Dabbous, 2001) as well as network packet routing (Sklower, 1991; Gupta, 2000), and semantic web database mapping acceleration (Blochwitz et al., 2016).

Patricia Trie consists of a root node, intermediate nodes, and edge nodes. The root node is the entry point for any input, it provides the first branching from the very beginning of the input. The edge nodes are the very end of the Trie and those have its corresponding filters. The intermediate nodes are the rest of nodes connecting links between nodes. Each node has a prefix filter of variable length (which can be alphabets, numbers, or any sequence of characters). The link connected to the next node is selected by the next character, i.e., the path holds a chain of nodes which constructs a cumulative prefix up until the node. To find a prefix or exact match of an input id, we traverse the tree from the root node and repeat the look-up until there is no more links from the node (reaches an edge node) or the cumulative prefix length become the same length of the id, as the further nodes from that node do not match with the input. The very difference between a Trie and Patricia Trie is that in a Trie, a node has single character regardless, but Patricia Trie concatenates the nodes with only one link, which makes any intermediate node in the tree have at least have two links with variable length filters.

FIGURE 4.4 shows the Patricia Trie structure for five ID patterns to filter. There are one root node, one intermediate node, and five edge nodes in this example. All the nodes are a subset of its parent (superior) node as they share a common cumulative prefix. For example, the edge node with “0101” is a subset of the node “1100” (11000101 1100).

From its nature, this algorithm is not suitable for finding a non-prefix pattern in an ID. When traversing the tree with a cumulative prefix and thus cannot choose the link if the intermediate path is blank or “wild-card” by its next bit (FIGURE 4.5). Even if we choose to move to the next node at an arbitrary position, since the wild-card filter can be variable length, the rest of the branch may end up matching with all the other filters as well. Even though Patricia Trie is not limited to a binary tree, intermediate nodes with a wild-card filter collapses the optimization for prefix matches.

Algorithms 4.3.2 and 4.3.3 show how to build a Patricia Trie, given a list of ID prefix patterns subscribed. The function *BuildNode* finds a pair of longest common prefix among the subscribed ID pattern for both cases; if it starts with 0 and 1. The longest common prefixes found then become the next hop node, and it recursively finds the next longest common prefix for each 0 and 1 case. As the node is placed deeper, the cumulative prefixes become longer. When searching the matched pattern of an ID with Patricia Trie, *Search* function traverses the node by selecting the links and trimming the matched prefixes from the ID. If the node is assigned with $notificationURI$ the ID is queued to report the ID to the corresponding subscriber.

FIGURE 4.6 shows the locality of each node in Patricia Trie when filtering a biased population of IDs with random configurations. Each circle represents a node, and its radius indicates the locality; the probability that an ID reaches the node. The tree shape is wide in horizontal manner, but the depth of the tree is maintained shallow to balance out the number of traversals for finding matches for an ID.

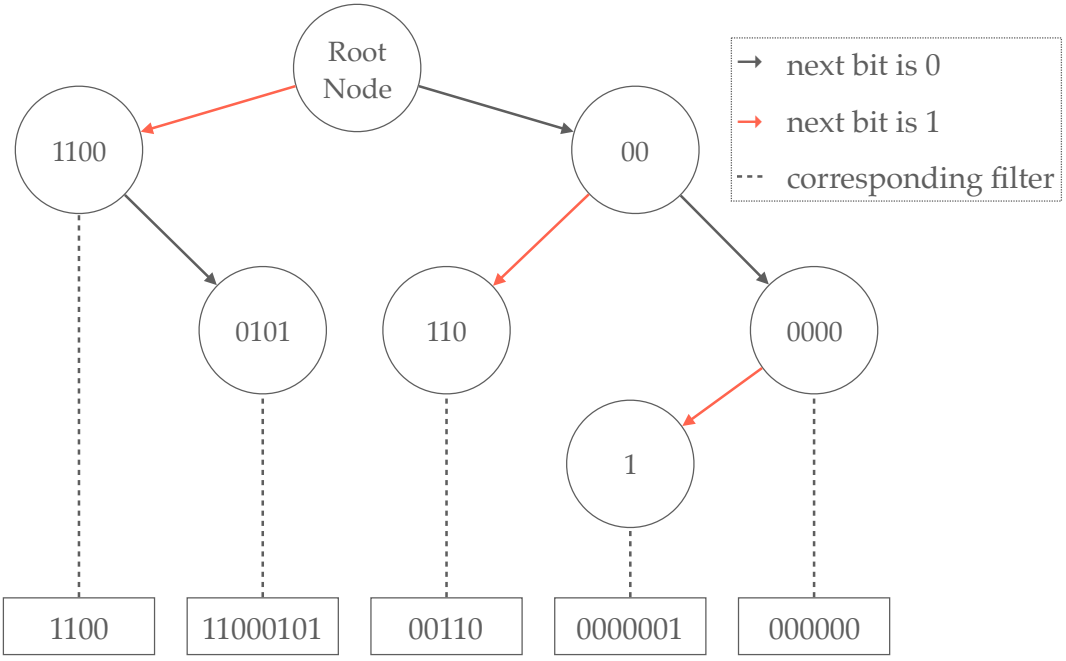


FIGURE 4.4: Patricia Trie of ID prefix filtering. The nodes only have two links at maximum since the IDs are in binary and the next possible bit is either 0 or 1.

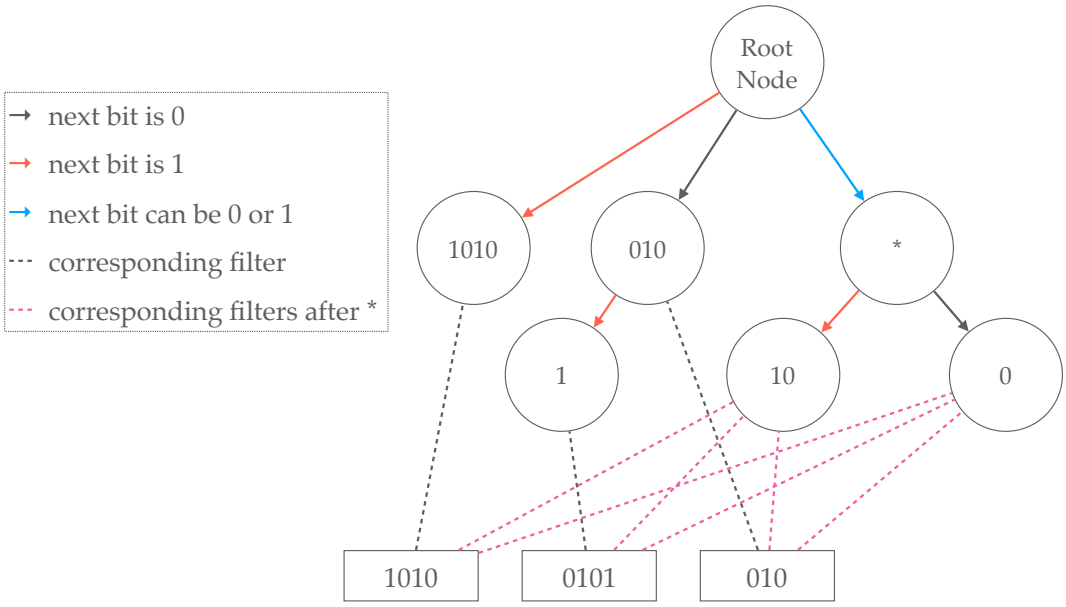


FIGURE 4.5: A blank intermediate node in Patricia Trie is invalid.

Algorithm 4.3.2. Patricia Trie: Main

Filters : A dictionary of value *notificationURI* by key *prefix*
compref: A string *compref* cumulative prefix from the root node
id : An ID to lookup
node : A pointer to the Node
prefix : A prefix pattern of IDs

```
1 Struct Node contains
2     Node zero;
3     Node one;
4     string prefix;
5     string notificationURI;
6 prefixes = [all prefix  Filters];
7 root = BuildNode(nil, "", prefixes);
8 Search(root, id);
```

Algorithm 4.3.3. Patricia Trie: Functions

```
1  Function BuildNode(node, compref, prefixes)
2      ZeroBranch ← string;
3      OneBranch ← string;
4      if node = nil then
5          node ← new Node;
6      node.filter ← compref;
7      if FILTERS has a key compref then
8          node.notificationURI ← FILTERS[compref];
9      for p ∈ prefixes do
10         if Length(p) ≥ compref or p not starts with compref then
11             Remove p from prefixes;
12             Next;
13         p ← Strip compref from p;
14         if p[0] = 0 then
15             if Length(ZeroBranch) = 0 then ZeroBranch ← p;
16             else ZeroBranch ← FindLongestCommonPrefix(p, ZeroBranch);
17         else if p[0] = 1 then
18             if Length(OneBranch) = 0 then OneBranch ← p;
19             else OneBranch ← FindLongestCommonPrefix(p, OneBranch);
20     if Length(ZeroBranch) = 0 then
21         node.zero ← BuildNode(node.zero, compref + ZeroBranch, prefixes);
22     if Length(OneBranch) = 0 then
23         node.one ← BuildNode(node.one, compref + OneBranch, prefixes);
24     return node
25 Function Search(node, id)
26     if id not match with node.filter then
27         return
28     if node.notificationURI is not empty then
29         Notify id to node.notificationURI
30     if (Length(node.filter) +1)th bit of id = 0 then
31         Search(node.one, TrimPrefix(id, node.filter))
32     else if (Length(node.filter) +1)th bit of id = 1 then
33         Search(node.zero, TrimPrefix(id, node.filter))
34     return
```

FIGURE 4.6: Visualization of locality per node in Patricia Trie. The radius of each indicates how high the locality of the node is (bigger if larger).

4.3.3 Optimal Binary Search Tree

Optimal Binary Search Tree (Optimal BST) filtering engine requires a specific condition: each filter needs to have its *weight*. The concept of Optimal BST is presented by Knuth, 1971 to build the best possible tree for searching keys with given probabilities. When all the filters in the tree are equally frequent to match IDs through the filtering engine, the Patricia Trie provides the minimum number of traversals to search matched patterns from the standpoint of average search time. However, the filters in the tree are not equally probable to have matching IDs. In fact, when F&C is connected to RFID readers on site, tra□c of IDs is skewed to make some filters to match more frequent and the others fewer. The locality of an arbitrary node then should be dynamic. The weight value could be simply the estimated amount of IDs that will match with the filter from the statistics or a even registered ID population from a logistics management system. In any case, the weight values must be provided to the filtering engine to use this algorithm. As the filtering engine knows how likely each filter will match with the input, it optimizes the order of filters so that frequent matches will capture the ID at an early look-up. On the other hand, it takes longer to capture minor IDs as the likelihood of the match with the corresponding filter is low. The di□erence between Linear Search algorithm is that it can branch out for subsets of a matched filter and reduce the redundant look-up for the rest of other filters. The look of Optimal BST is a tree with long one-arm tree with branches of subsets, and it tends to be deeper and narrower than Patricia Trie. FIGURE 4.7 shows the Optimal BST for the same filters as in FIGURE 4.4, but the weight values are given as in TABLE 4.1.

TABLE 4.1: The weight values for the five filters in the example.

Filter	Weight value	Probability
000000	100	0.385
1100	80	0.308
0000001	50	0.192
11000101	20	0.077
00110	10	0.038

The algorithm to build an Optimal BST and to search prefix matches with an ID is shown in Algorithm 4.3.4.

FIGURE 4.8 shows the locality of each node in Optimal BST. Unlike the Patricia Trie (FIGURE 4.6), it has many subsets under the nodes, and the shape results in deeper as the locality of the node decreases. Higher the node positions, the locality of the node becomes higher as it is built in order.

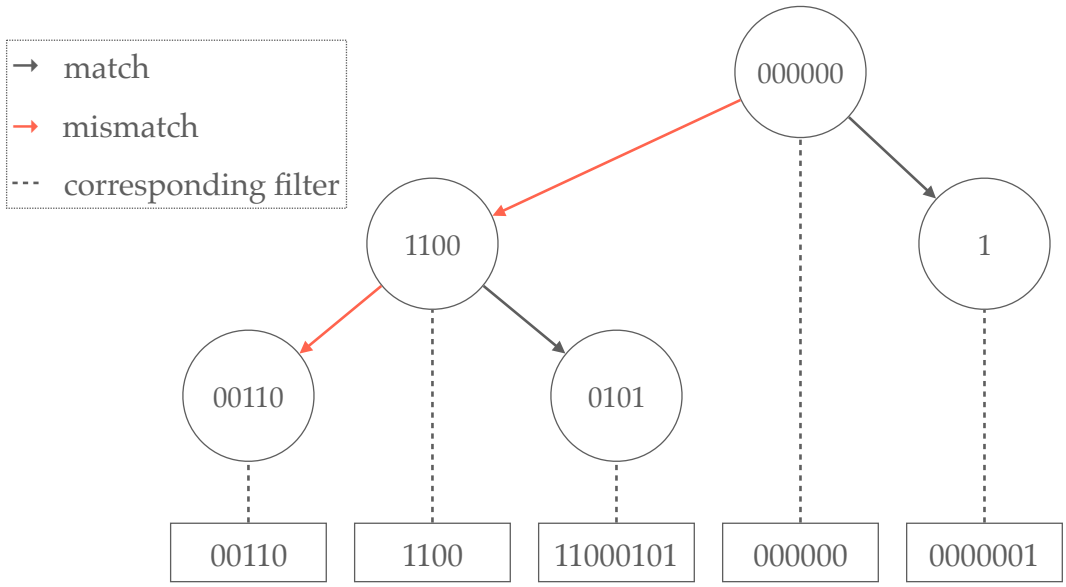


FIGURE 4.7: Optimal Binary Search Tree (Optimal BST) with five prefix filters. The nodes have two links: match or mismatch.

Algorithm 4.3.4. Optimal Binary Search Tree

```
WVMap: A dictionary of prefix as key, weightValue as value
Filters : A dictionary of value notificationURI by key prefix
id      : An ID to lookup
node    : A pointer to the Node
prefix  : A prefix pattern of IDs
Struct Node contains
    Node matched;
    Node mismatched;
    string prefix;
    string notificationURI;
prefixes = [all prefix  WVMap in descending order of weightValue];
n ← new Node;
root ← n;
for i ← 0 to Length(prefixes) do
    n.prefix ← prefixes[i];
    n.notificationURI ← FILTERS[prefixes[i]];
    for j ← 0 to Length(prefixes) do
        if prefixes[i] = prefixes[j] then
            Next;
        if prefixes[j]  prefixes[i] then
            if n.matched = nil then n.matched ← new Node;
            BuildSubset(n.matched, prefixes[j]);
            Remove prefixes[j] from prefixes;
    n.mismatched ← new Node;
    n ← n.mismatched;
Search(root, id);
Function BuildSubset(node, prefix)
    if Length(node.prefix) = 0 then
        node.prefix ← prefix;
        return;
    if n.mismatched = nil then n.mismatched ← new Node;
    ;
    BuildSubset(n.mismatched, prefix);
    return;
Function Search(node, id)
    if id matches with node.filter then
        Notify id to node.notificationURI;
        if n.matched = nil then
            Search(node.matched, id);
    Search(node.mismatched, id);
    return;
```

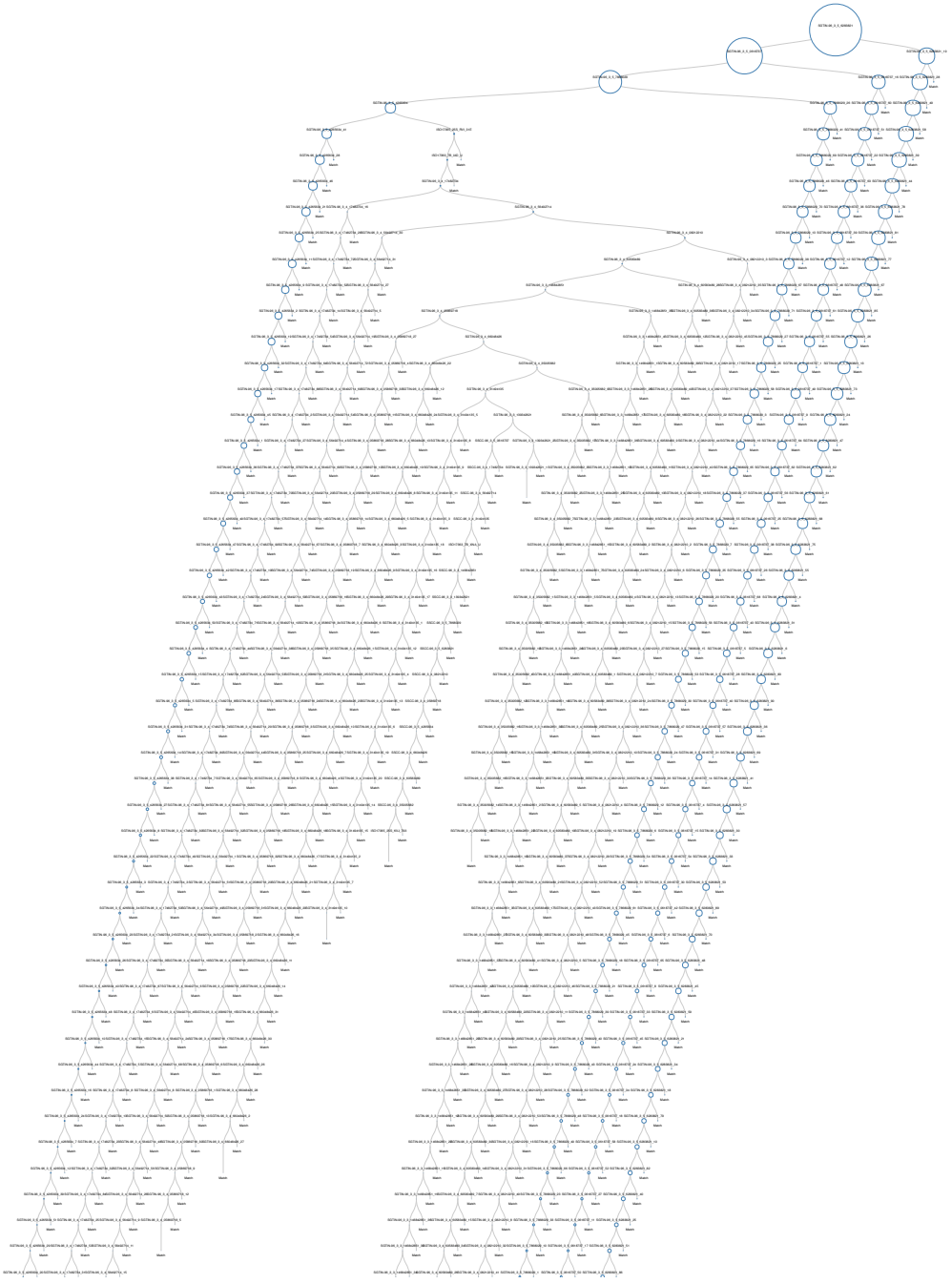


FIGURE 4.8: Visualization of locality per node in Optimal BST. The radius of each indicates how high the locality of the node is (bigger if larger).

4.3.4 Splay Tree

Splay Tree is similar to Optimal BST in terms of the structure of the tree, and in fact it can be implemented by modifying the *Search* function in Optimal BST and make it a wrapper for a new function *SplaySearch*. The snippet of the algorithm is shown in Algorithm 4.3.5. The difference from Optimal BST is that it doesn't require the *weight* value, and it reorders the nodes in the tree every time the *Search* function is called. At the end of each search, Splay Tree replaces the node that has matched the ID with the entry node of the tree, as a consequence, the last matched node comes as the first node in the next search. By repeating the procedure, the tree converges to the state that is optimized for the trend of the ID traffic in the past. The performance of this algorithm can be affected by the order of IDs processed in the filtering engine. Whereas Splay Tree outperforms to filter IDs with frequently observed patterns, it is definitely not suitable for a situation of finding exact matches unless the IDs are expected to be read many times.

Algorithm 4.3.5. Splay Tree (snippet)

```
1  Struct SplayNode contains
2      Node root;
3  spn ← new SplayNode;
4  spn.root ← new OptimalBST;
5  Search(spn, id);
6  Function Search(spn, id)
7      SplaySearch(spn.root, id, nil, spn);
8      return;
9  Function SplaySearch(node, id, parent, spt)
10     if id matches with node.filter then
11         Notify id to node.notificationURI;
12         if n.matched = nil then
13             SplaySearch(node.matched, id, node, spt);
14         if parent = nil then
15             parent.mismatched = node.mismatched;
16             node.mismatched = spt.root;
17             spt.root = node;
18     SplaySearch(node.mismatched, id, node, spt);
```

4.3.5 Filtering Execution

In addition to the search algorithms introduced in the previous clauses, ID filtering by byte-wise comparison is highly preferable instead of performing string comparison. Conversion of the subscribed filters into binary format is necessary beforehand but it significantly contributes to the filtering throughput. The minimum possible size of a filter is 1 bit, and we need to pad the bits that are not of the interest when performing the byte-wise comparison at the boundary of bytes (multiple of 8-bits from the beginning). The minimum size of variables in modern computing platform is 8 bits; hence, the minimum length of a filter is also 8-bit long. For a filter of the length of 1 byte, the possible patterns are as shown in FIGURE 4.9.

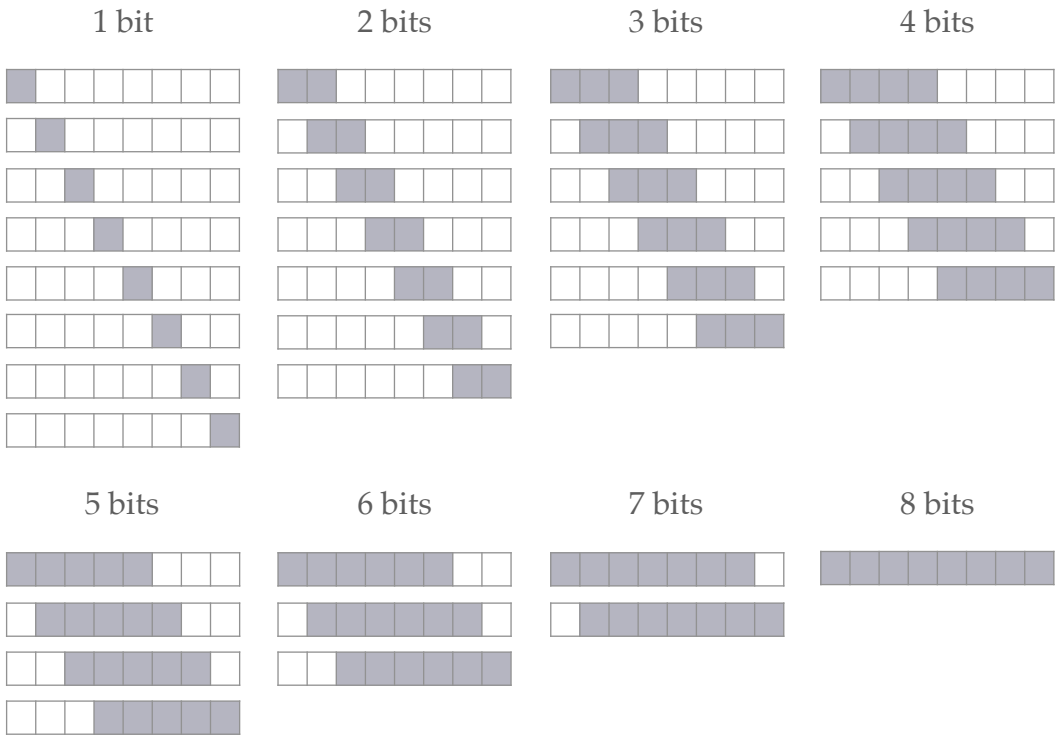


FIGURE 4.9: Bit-Filter (shorter than 1 byte) Patterns.

If the filter has more than 1 byte and it continues beyond the byte boundary, the possible patterns are as shown in FIGURE 4.10. The patterns from FIGURE 4.9 falls into either of a, b, c, or d of this classification, and for the filters with more than 1-byte length can be expressed by a combination of those.

The performance gain of byte-wise comparison rather than string comparison in my Go implementation is shown in TABLE 4.2. Depending on the use cases, the throughput of the byte-wise filtering process became almost twice to five times faster than the string comparison implementation.

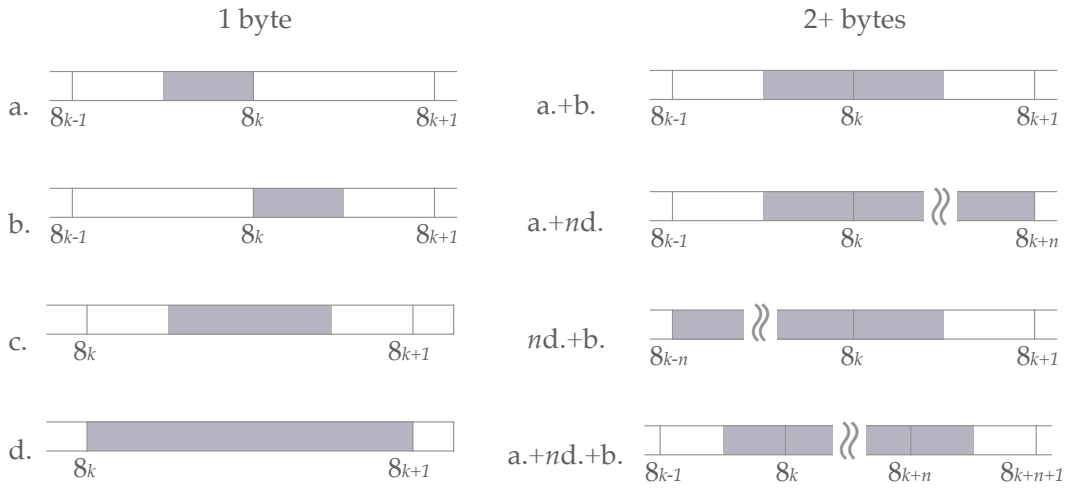


FIGURE 4.10: Byte-Filter (longer than 8 bits) Patterns.

TABLE 4.2: The effect of byte-wise comparison to eliminate string comparison process.

	5,600 IDs & 63 Filters	15,915 IDs & 4,763 Filters	32,000 IDs & 264 Filters	320,000 IDs & 2,640 Filters
String	50 ms	3,300 ms	630 ms	25,370 ms
Byte-wise	10 ms	1,100 ms	150 ms	13,030 ms

Chapter 5

Evaluation of the Filtering Engine

This chapter evaluates the filtering engines based on the four algorithms introduced with each scenario from Scenario 1, Scenario 2, and Scenario 3.

5.1 Scenario 1 – A container terminal

The characteristic of Scenario 1 is the heavy tra□c of IDs (10,000 IDs/sec) and the registered filters are relatively shorter, to capture the ownership of the SGTIN the prefix before the Company Prefix is su□cient, for example. So the dataset is prepared to simulate 148825 multi-code IDs and 721 filters of shorter filters, and FIGURE 5.1 shows the time it took for each filtering engines to build and process the ID filtering. For Optimal BST, the order of IDs to be streamed to the filtering engine has a significant impact, I randomized the order of IDs in comparison to the stream of IDs in descending order. FIGURE 5.2 shows the histogram and the distribution density estimation of the subscribed filters for how many IDs actually matched. The majority of the filters match with less than 100 IDs, but as a whole it is a long-tail to the extent of the filters matching more than 40,000 IDs. This is because the dataset is configured to have administration level filtering such as the filtering by Filter Value in GS1 RF tags or even the standard that RF tags are encoded. From the result, Splay Tree achieved the reasonable performance throughput with the 500 ms of building the engine, which is almost 10% for the Patricia Trie engine to be built. To support the result, I performed additional simulations by keeping the ratio of ID/Filter the same, and modified the size of the dataset. FIGURE 5.3 shows the performance order remains similar regardless of the size of dataset, except for the build time for Patricia Trie increases. For the scenario like the terminal use case, it is still acceptable for longer building time since the subscription frequency is low, and Splay Tree maintains the required throughput at the best among other algorithms.

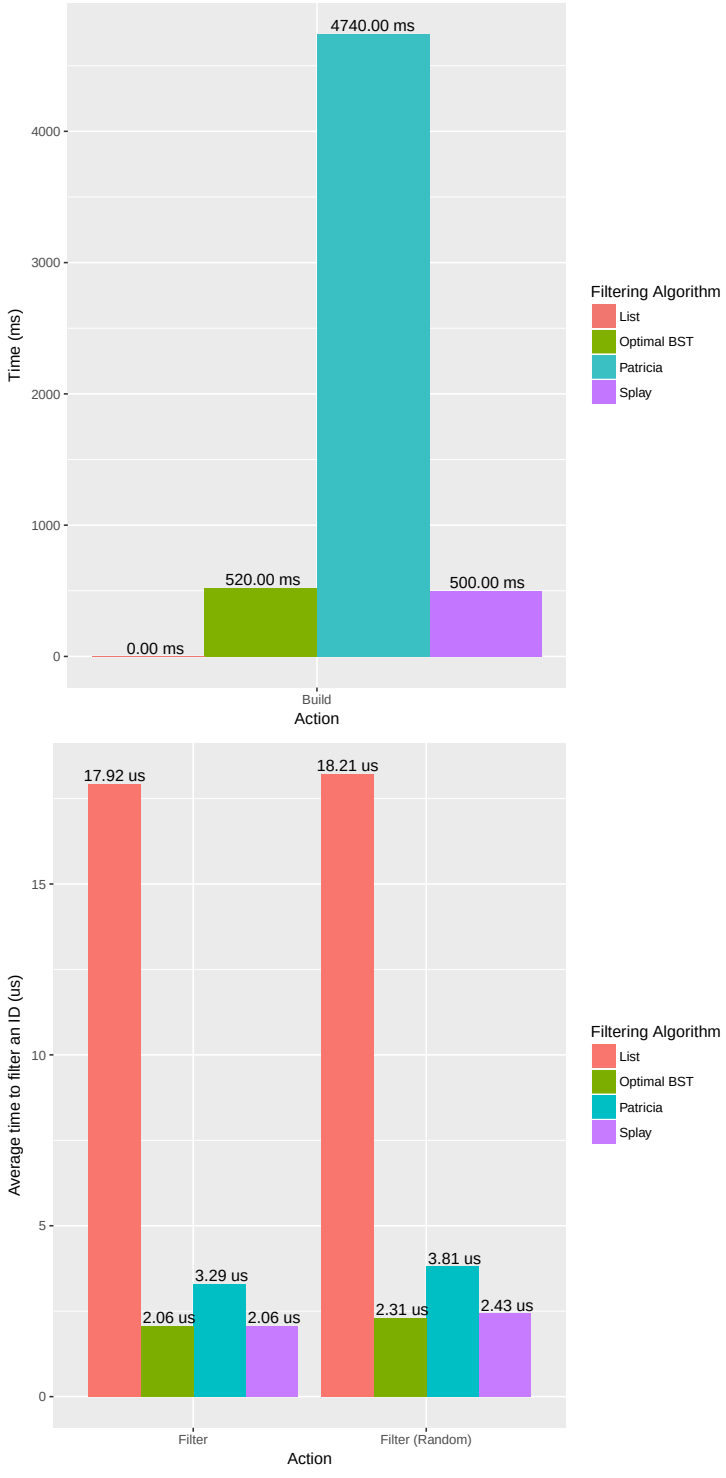


FIGURE 5.1: Simulation Scenario 1: 148825 IDs / 721 Filters.

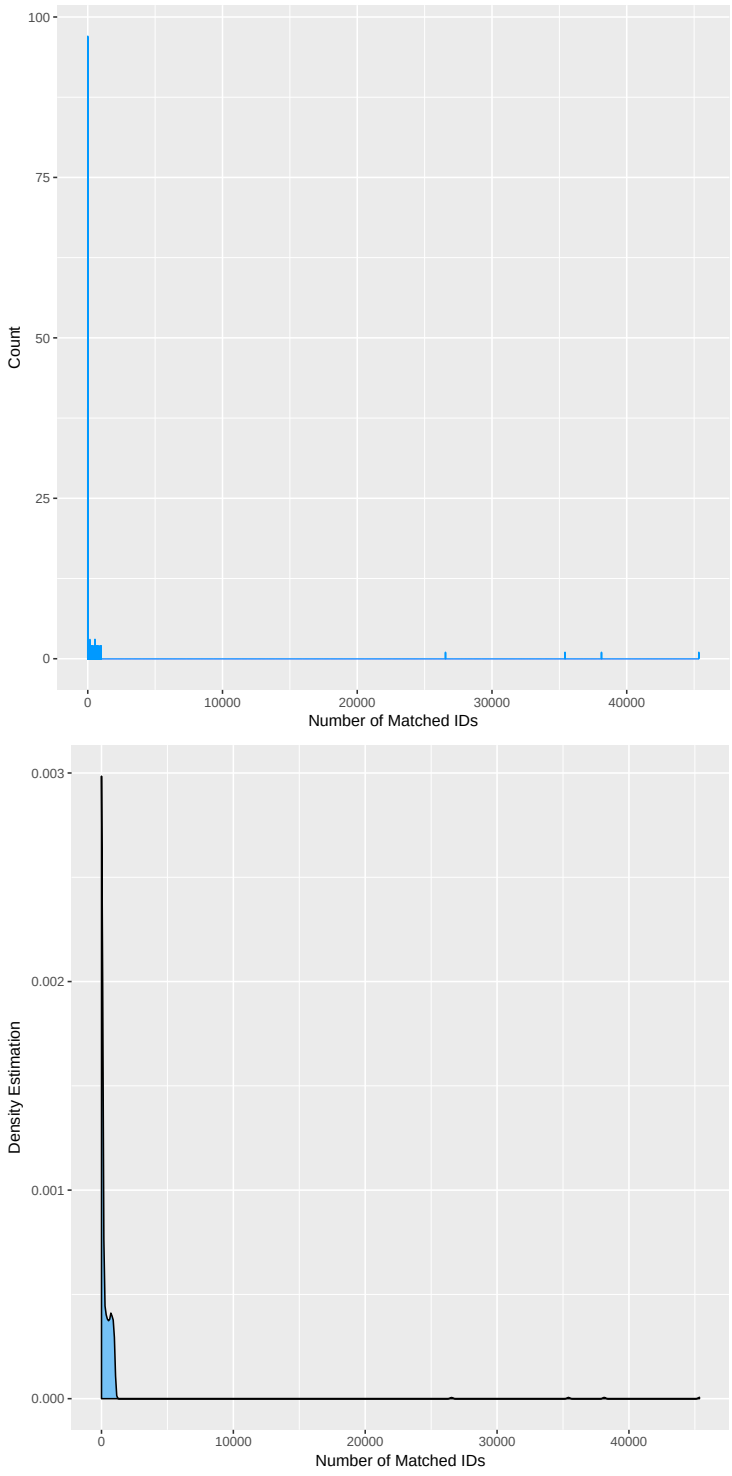


FIGURE 5.2: Histogram and Kernel density estimation of the subscribed filters in Scenario 1 dataset (148825 IDs / 721 Filters).

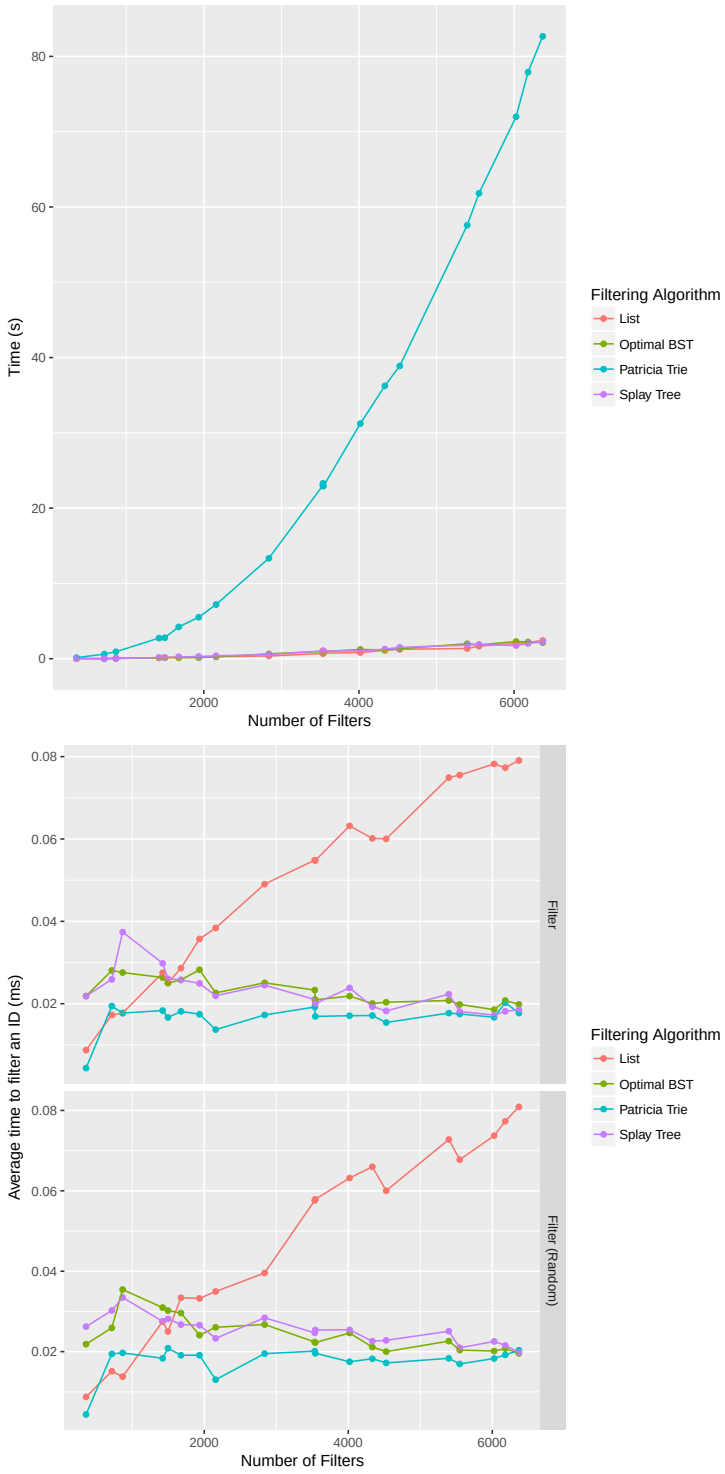


FIGURE 5.3: Performance result for Scenario 1 per filtering algorithm with different dataset. The plots in Filter groups indicate the time taken to filter an ID (Second/ID).

5.2 Scenario 2 – A convenience store

The characteristic of Scenario 2 is high-frequent subscription/subscription rate of the filters. As the filtering engine needs to be rebuilt at every POS events from the checkout system, the time required for rebuilding the engine is critical. The dataset is prepared to simulate the realistic population of items in convenience stores, assuming 2800 items with 1-100 stocks in-store, 44456 SGTIN IDs and 47561 filters. FIGURE 5.5 is the result plot of the filtering engine performance from the data set, as it took incomparably long time for Patricia Trie to build from the number of filters, Patricia Trie should be excluded from the optimization candidate. FIGURE 5.6 shows the distribution density of the subscribed filters for the dataset, indicating all the filters are the "exact matching" filters including serials for individual item management for auto-checkout systems. Obviously for this case Patricia Trie is not an option for the filtering engine, both Optimal BST and Splay Tree take almost 200 seconds to build the engine, while List only takes 10 seconds. FIGURE 5.7 is the result on the similar dataset representing the smaller number of filters and items at a given time, by considering the first dataset as the worst case. For the convenience scenario, Patricia Trie is not usable to catch up with the frequent subscription rate, List is the only option. The ID processing throughput of List algorithm is much smaller than the others, but it is still admissible in the context of shopping at the POS automatic checkout system (approximately 1 ms for an ID to be processed).

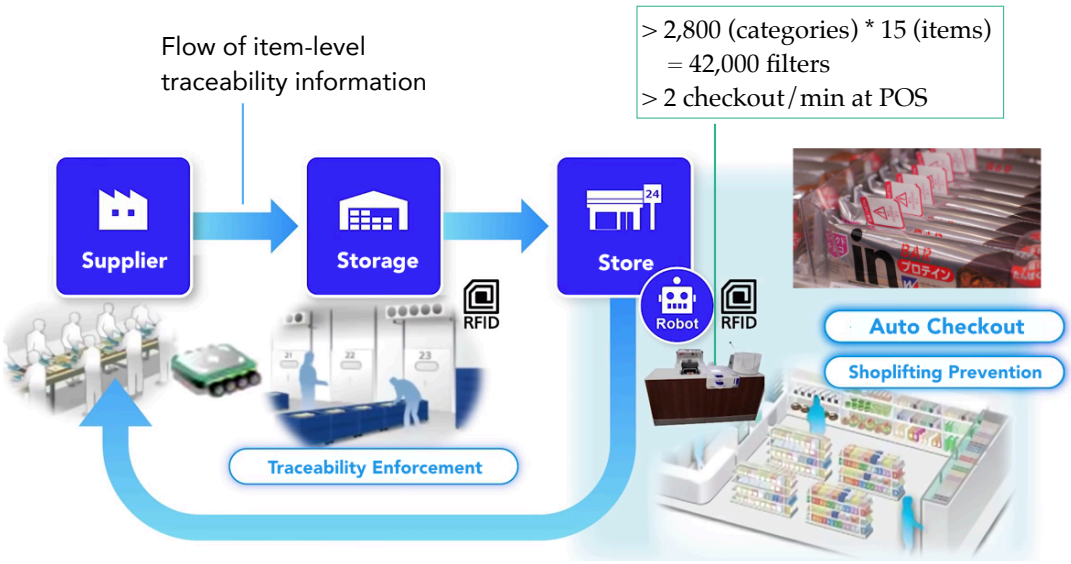


FIGURE 5.4: Overview of Scenario 2. Part of the image belong to Panasonic (Robotic Checkout System "Regi-robo(TM)" with RFID Tags for Next Generation of Retail: https://channel.panasonic.com/contents/20044/).

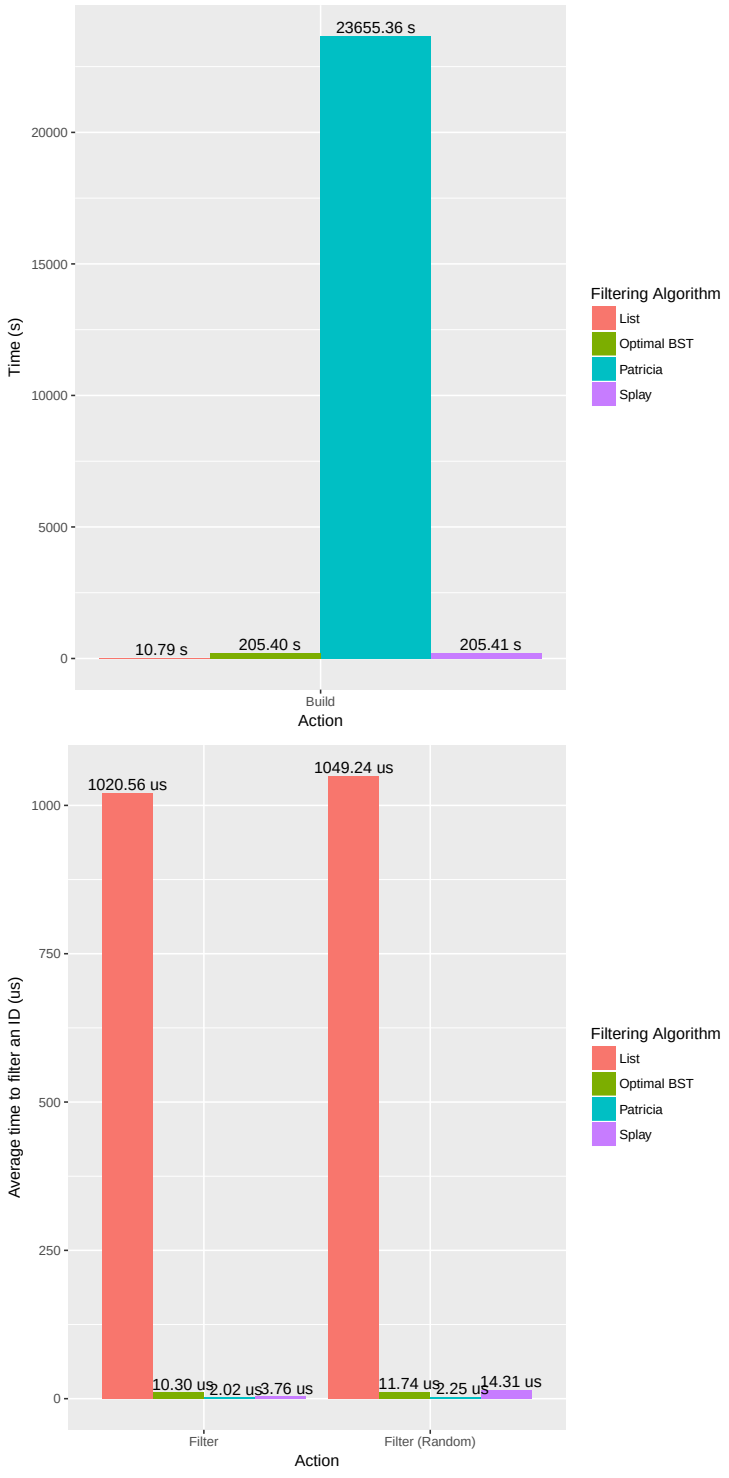


FIGURE 5.5: Simulation Scenario 2: 44456 IDs / 47561 Filters (Patricia Trie is excluded due to the exponentially long build time)

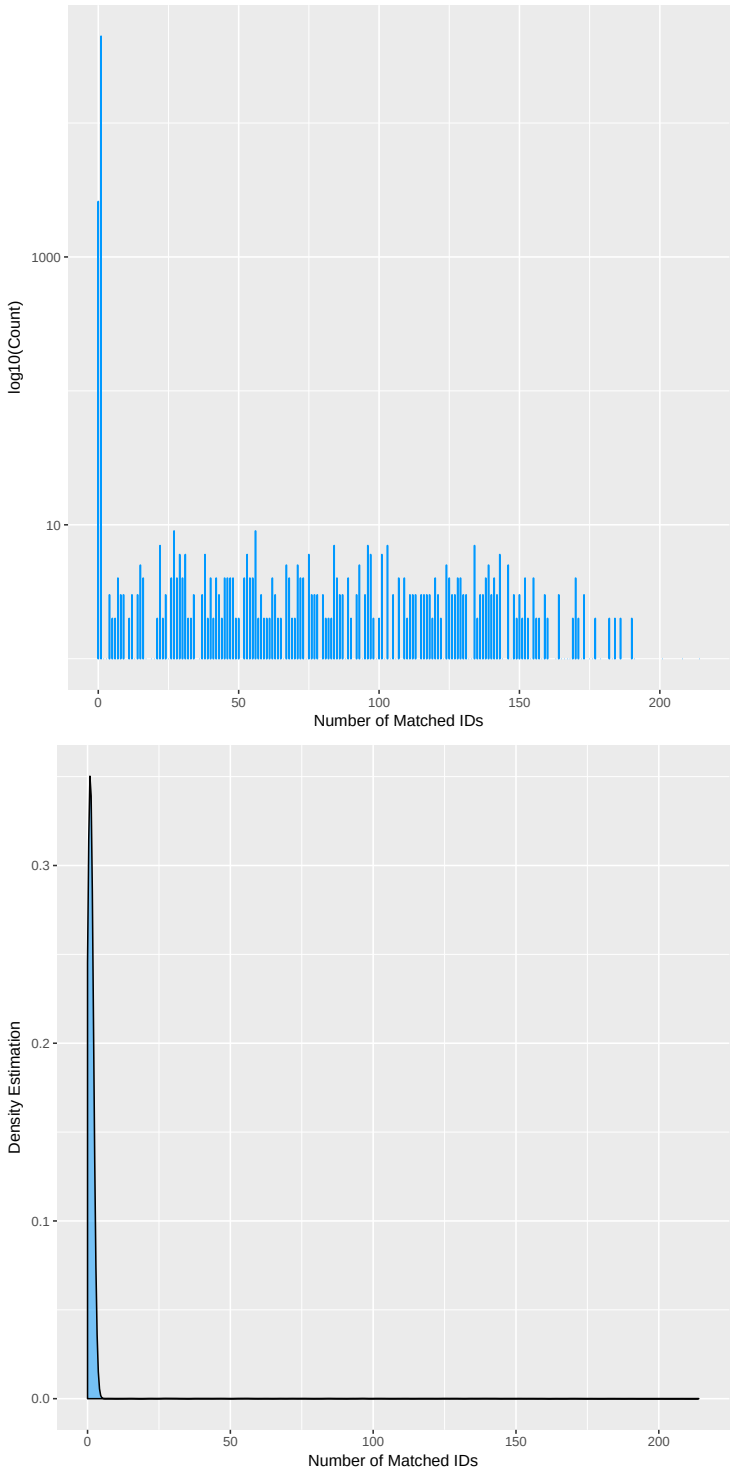


FIGURE 5.6: Histogram and Kernel density estimation of the subscribed filters in Scenario 2 dataset (44456 IDs / 47561 Filters).

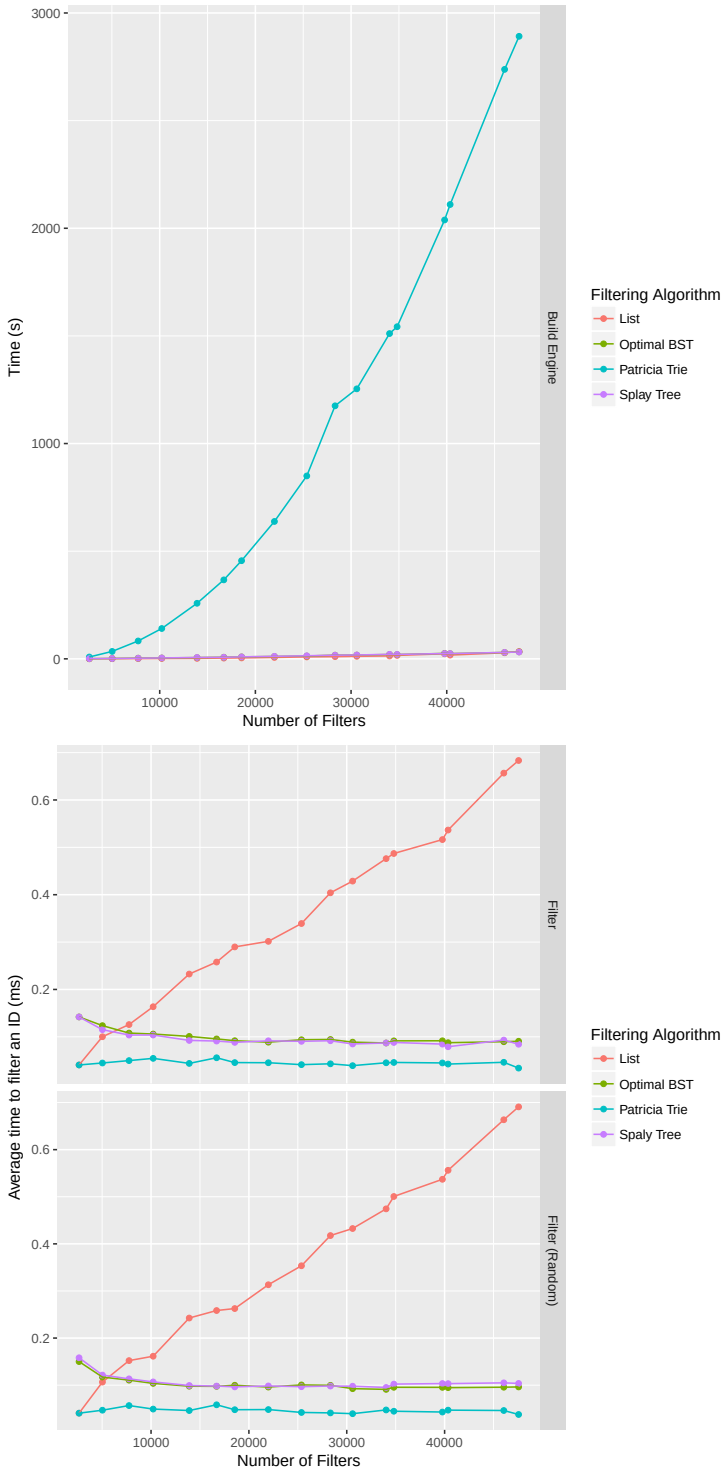


FIGURE 5.7: Performance result for Scenario 2 per filtering algorithm with di□erent dataset. The plots in Filter groups indicate the time taken to filter an ID (Second/ID).

5.3 Scenario 3 – A shopping mall

The distinctive characteristic of Scenario 3 from the previous scenarios is that the filtering engine doesn't have the estimated tra□c of IDs. In the shopping mall, in addition to the items sold in the next door, the customers bring in the items with RF tags that have been purchased from the other stores in the mall, and the amount of those tags could be enormous. The dataset for this scenario is prepared so as to generate an intermediate amount (and the ratio) of IDs and filters between the previous two scenario dataset. FIGURE 5.9 shows the result for this dataset, and FIGURE 5.10 is the distribution density of the subscribed filters per the matched number of IDs. The ID filtering performance of Patricia Trie is superior with the feasible build time. Since the subscription frequency is not as important as in Scenario 2, less than 5 minutes of the engine build time doesn't a□ect the retail store inventory management operation. To achieve the flawless shoplifting detection, the ID processing throughput weighs more than the build time. For both Optimal BST and Splay Tree, an ID needs to go through the longest traversal of the tree if the ID doesn't match any filter. Patricia Trie is the most preferable option for this scenario.

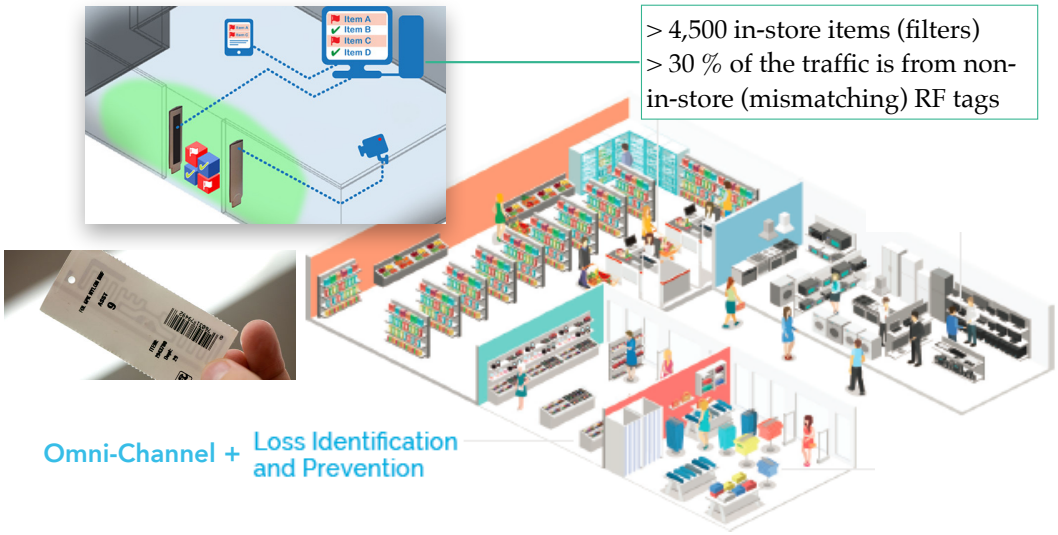


FIGURE 5.8: Overview of Scenario 3. Part of this image belong to firefly RFID solutions (Adding RFID to EAS Systems: http://fireflyrfidsolutions.com/industry-solutions/retail/adding-rfid-to-eas-systems/index.html) and UST Global (RFID SUCCESS STRATEGIES FOR APPAREL RETAILERS: https://www.ust-global.com/retail-rfid-success-strategies-apparel-retailers).

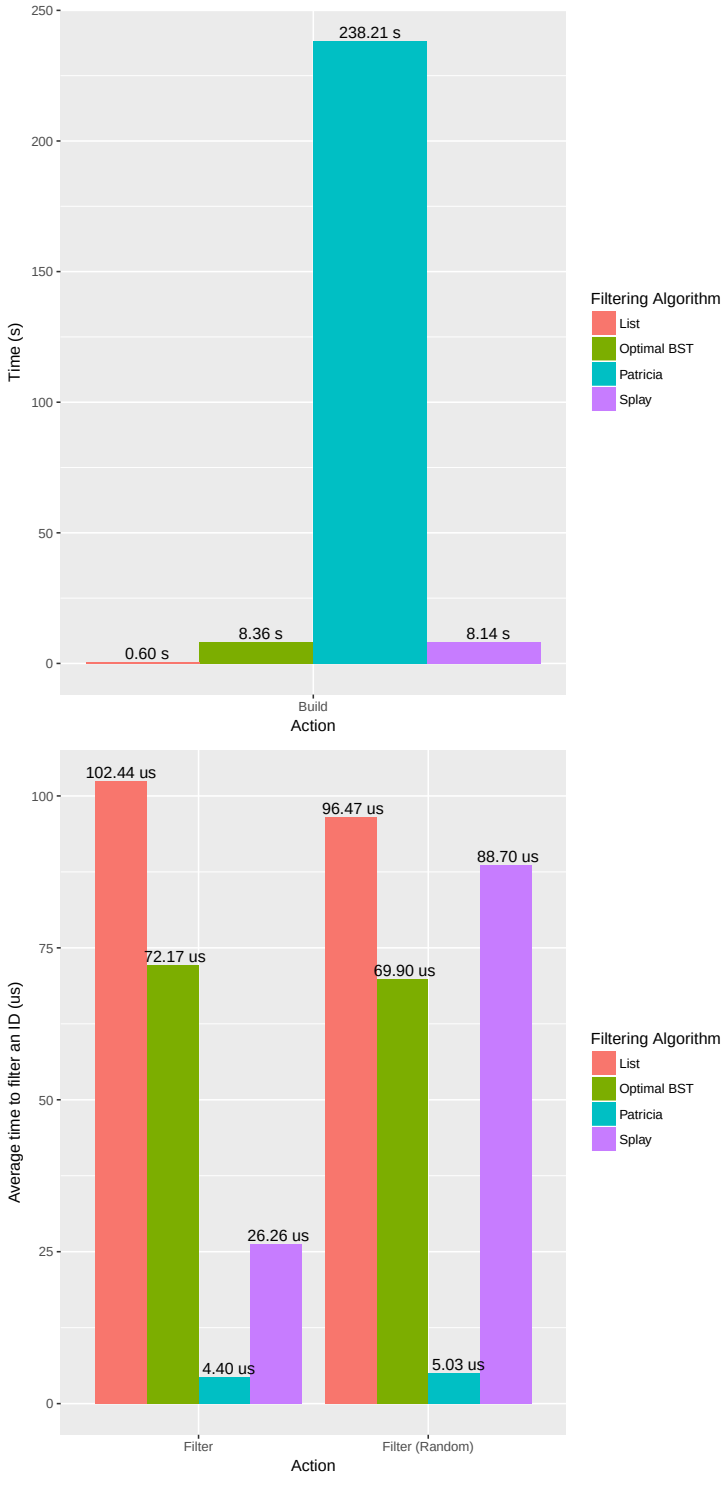


FIGURE 5.9: Simulation Scenario 3: 15915 IDs / 4763 Filters

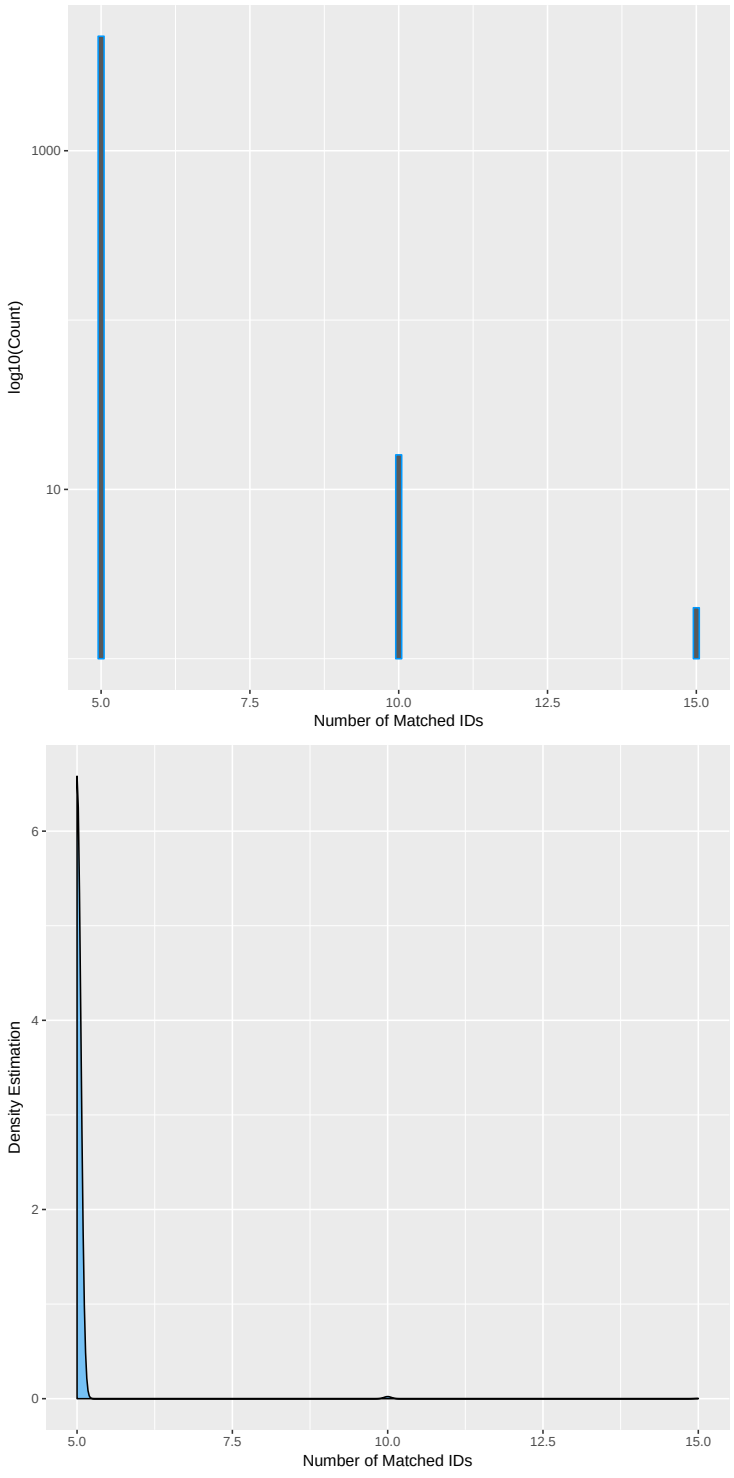


FIGURE 5.10: Performance result for Scenario 3 per filtering algorithm with different dataset (15915 IDs / 4763 Filters).

5.4 Integrity of ID Pattern Matching

The ultimate goal of the algorithm for filtering IDs encoded in RF tags is to determine if an arbitrary binary string matches with a subscribed pattern. The pattern is, most of the case, prefix of the IDs as described in Chapter 2; however, in some cases, somewhere in the middle of the binary string could become one's interests (non-prefix pattern).

Although the demands for that kind of requests is considered to be low as it violates property rights beyond owners as mentioned in Chapter 3, the possibility is not negligible at the operational sites. Logistics or sales operators might be interested in capturing all the items with a specific packaging level to grasp the operational flow. For instance, *GTIN-14* (which can be a part of SGTIN-96 or SGTIN-198) has a *Indicator* digit to identify the purchase unit for consumers (packaging level) in front of *Item Reference* element. Likewise, the operators might also want to know the distribution of ISO 17365 IDs by *Issuing Agency Code*. They have a filtering element at an arbitrary position of the ID and the other part can be anything (FIGURE 5.11,5.12).

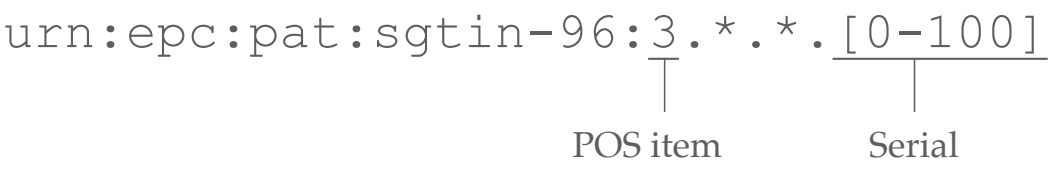


FIGURE 5.11: Non-prefix ID pattern example: this pattern indicates any ID of POS that has a serial between 0 and 100. The ID can have any Company Prefix and Item Reference.

FIGURE 5.12: Non-prefix ID pattern example: this pattern indicates any SGTIN-96 ID of Company Prefix 458960468 (Auto-ID Lab. Japan). The ID can have any Filter value, Item Reference, or Serial.

Allowing this kind of filters to exist in the filtering engine makes its structure to the whole different level of difficulty rather than prefix matches. To maintain the filtering engine's integrity, a conceptual Venn diagram can be drawn as FIGURE 5.13 to guarantee any IDs to be properly captured by all of corresponding filters. In the diagram, it is assumed that there are six different filters. $Filter_A$, $Filter_B$, and $Filter_C$ have a common pattern with each other and $Filter_F$ is a subset of $Filter_E$ which have no common pattern from the first three filters.

Each circle indicates the subsets of IDs matching with the corresponding filter and the number indicates a region of its color (an independent subset of the filter). There are three kinds of regions:

1. A region belongs to a single filter or the region itself is a filter: *region* $Filter_i$. (e.g., 1, 3, 5, 9, 12)

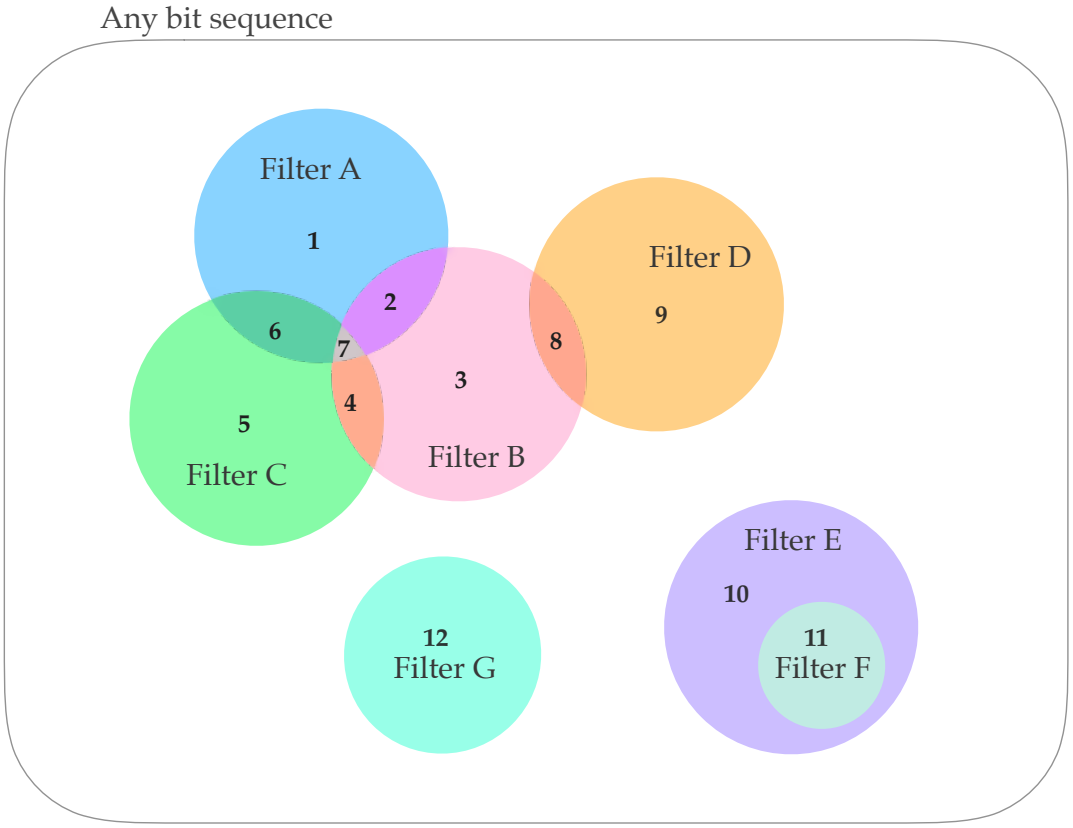


FIGURE 5.13: Integrity of ID pattern filtering in Venn Diagram.

2. A region belongs to multiple filters: $region \quad Filter_i \cap region \quad Filter_j$. (e.g., 2, 4, 6, 7, 8, 10)
3. A region belongs to multiple filters and the region itself also represents a filter: $region \quad Filter_i \cap region \quad Filter_j$. (e.g., 11)

FIGURE 5.14 depicts how the filtering engine should classify an ID and aggregate resulting groups to generate reports for each filter. For example, there are four cases when an ID matching with $Filter_A$ ($= \{1, 2, 7, 8\}$) handed over to the filtering engine.

- ID matching with only $Filter_A$
- ID matching with both $Filter_A$ and $Filter_B$, but $Filter_C$
- ID matching with both $Filter_A$ and $Filter_C$, but $Filter_B$
- ID matching with all $Filter_A$, $Filter_B$, and $Filter_C$

All the four types of IDs should be reported to the subscribers of $Filter_A$.

The aggregated IDs matched with each filter will then be mapped with designated subscribers and will be reported.

Although non-prefix patterns are theoretically subscribed in the middleware, filters in practice the prefix are obviously the vast majorities because of the hierarchical structure of IDs. Rather than considering minor cases, I focused on performance tuning on the scenarios proponed in Chapter 3.

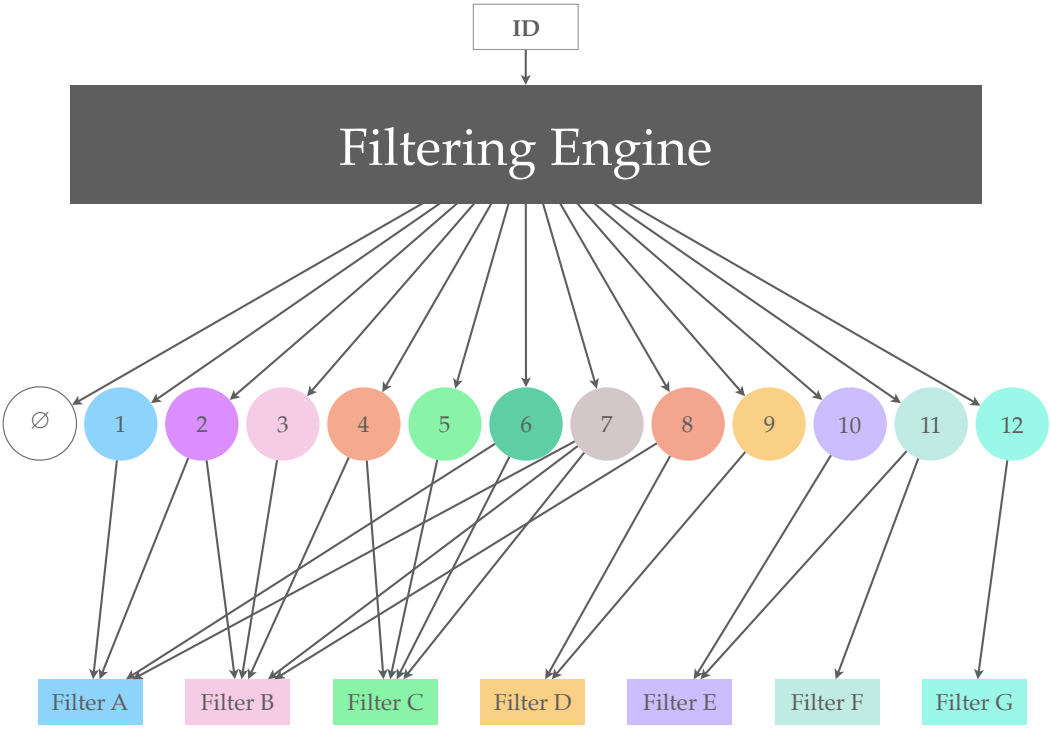


FIGURE 5.14: An ideal filtering engine.

Chapter 6

Techniques for Practical Deployment

The middleware should be capable of meeting the required throughput and load-balance architecture to fully utilize instantiated filtering engines. This chapter makes supplementary suggestions of techniques to further improve the robustness of the middleware.

The modular design maintains the dependence properties at minimum, such as the waiting time for an ID to be processed by the filtering engine or aggregation of matched IDs for generating reports. Automatic parallelization of the filtering engine is realized by the stochastic model along with Queuing Theory. The determination factors are the filtering rate, the traffic of IDs (ID/s), and the amount of possible computational resources; e.g., under-utilized CPU cores, unused memory space, or access to deployment environment in Cloud. For robustness, state-of-the-art cloud computing platforms provide elastic deployment environment for the middleware by virtualization technologies.

6.1 Parallelization and Horizontal Scaling

In this section, I discuss the effort toward the parallelization of the middleware and the method to estimate the number of parallel filtering engines by analyzing the rate of ID filtering in the filtering engines with Queuing Theory.

6.1.1 Local and Remote Parallel Filtering

To resolve the delay propagation issue mentioned in Clause 3.2.1, I designed the middleware to be able to spawn as many filtering engine component as necessary to catch up with the required throughput. (Chapter 4) The parallelization for both local (on the same machine as multiple process) and remote (on a different machine, VM, or Docker container) can be realized as long as the filtering engine can be transmitted. gosstrak-fc implements each filtering engine with Marshaller and Unmarshaller to transform the memory representation of the filtering engine to a data format suitable for duplication and transmission. The size of filtering engine varies by the algorithm. FIGURE 6.1 shows the transition of the size of filtering engine as per the number of filters consist of.

Even for local parallelization, memory copy of the filtering engine is recommended for a thread-safe access to the memory addresses. The internal protocol between

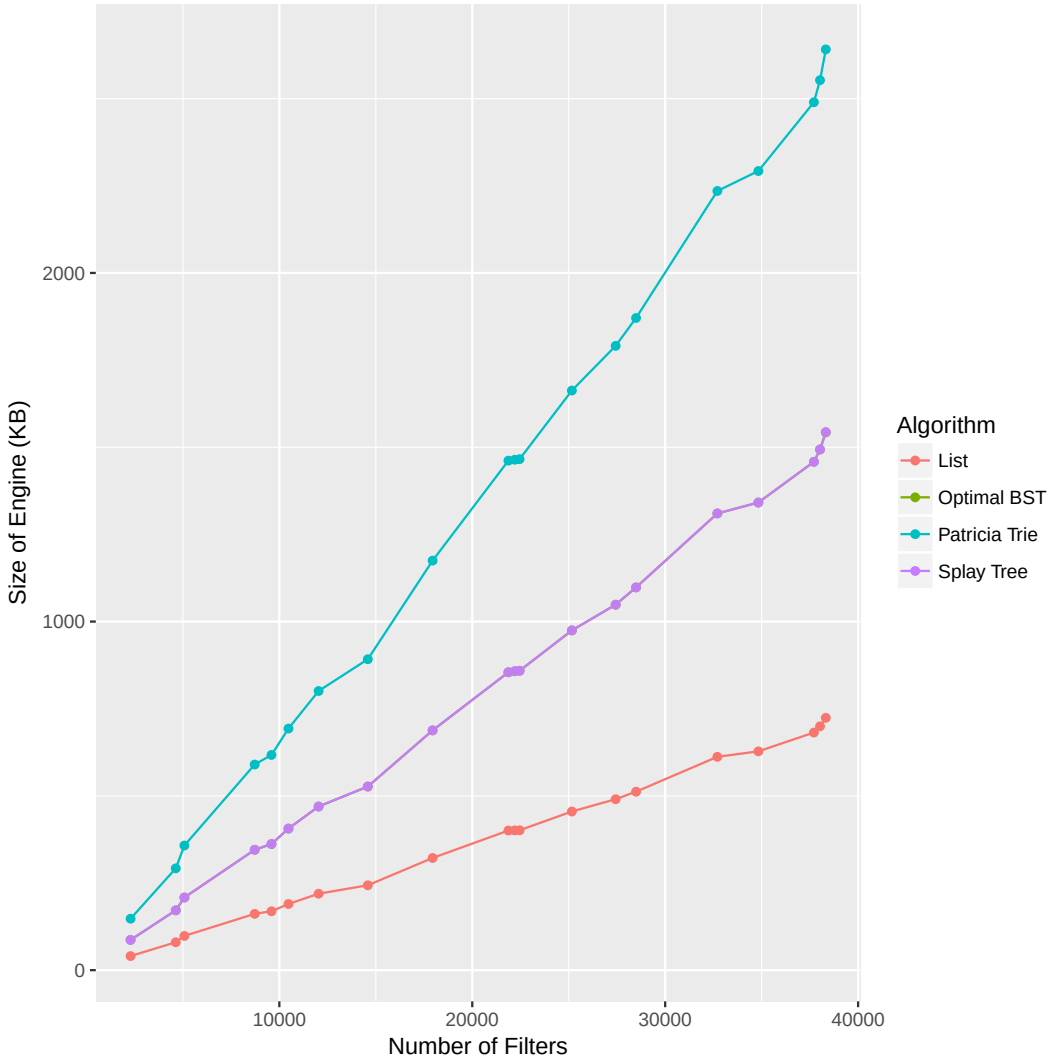


FIGURE 6.1: Size of the filtering engine by the number of filters.

gosstrak-fc filtering engine nodes is designed with gRPC[1] so that it can be inter-communicated with other software.

6.1.2 Analysis with Queuing Theory

If we assume the rate of RF tags read by the readers connected by the middleware follows Poisson distribution, the time it takes to filter the group of IDs with n filtering engines is stochastic. The state of the middleware in terms of n filtering engines has *Markov property* $X(n), n \quad \mathbb{N}$ for all times $n \quad \mathbb{N}$ and for all states $(i_0, i_1, \ldots, i_n)$ as we know the stochastic throughput for each filtering engine from Chapter 5.

$$P\{X_n = i_n | X_0 = i_0, i_1, \ldots, i_n\} = P\{X_n = i_n | X_{n-1} = i_{n-1}\} \quad (6.1)$$

[1] https://grpc.io/

As far as the present state of the system (e.g., the estimated tra□c of IDs) is provided, we are able to make predictions about its future state (how long it should to take to finish the queue) without counting on previous states. When an ID reaches one filtering engine, the ID is queued if the engine is still occupied by another ID. Queuing process is often explained by a system that consists of customers (IDs) arriving for service (filtering), waiting in a queue for the service if necessary, and after being served, leaving the system (reported to the subscribers).

In Queuing Theory by Kleinrock, 1975, the most fitted case for the estimation of p_k with *ma* filtering engines is M/M/m queue, the *m*-server case. FIGURE 6.2 shows the state space diagram for M/M/m queue, where *c* is the parameter for *m* with the symbols defined in TABLE 6.1

TABLE 6.1: Symbols in the M/M/m State Space Diagram.

Symbol	Definition
λ	Average read rate in IDs/sec
μ	Average filtering rate in IDs/sec
m	The number of filtering engines
ρ	The utilization of the filtering engines (IDs are queued when $\rho \geq 1$)

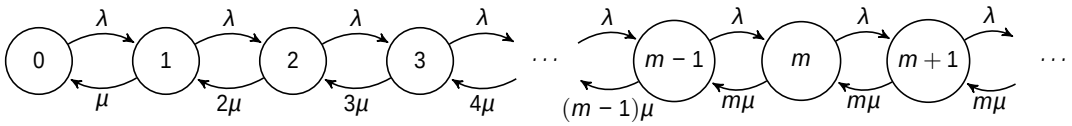


FIGURE 6.2: M/M/m State Space Diagram.

To estimate the most e□cient number of filtering engines to deploy, we first solve p_k for $k \leq m$.

$$
\begin{aligned}
p_1 &= \frac{\lambda_0}{\mu_1} p_0 \\
p_2 &= \frac{\lambda_1}{\mu_2} p_1 = \frac{\lambda_0 \lambda_1}{\mu_1 \mu_2} p_0 \\
p_k &= \frac{\lambda_0 \lambda_1 \cdots \lambda_{k-1}}{\mu_1 \mu_2 \cdots \mu_k} p_0 \\
&= p_0 \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}} \\
&= p_0 \left(\frac{\lambda_0}{\mu_1} \right)^k \frac{1}{k!} \qquad (0 \leq k \leq m)
\end{aligned}
$$

μ_i becomes a constant ($m\mu$) for $k \geq m$ (6.2) and the filtering process capacity will not increase as the ID grows.

$$\mu_k = m\mu_1 \qquad (m \leq k) \tag{6.2}$$

and therefore,

$$\begin{aligned} p_k &= p_0 \prod_{i=0}^{m-1} \frac{\lambda_i}{\mu_{i+1}} \prod_{j=m}^{k-1} \frac{\lambda_j}{\mu_{j+1}} \\ &= p_0 \left(\frac{\lambda_0}{\mu_1}\right)^m \frac{1}{m!} \left(\frac{\lambda_0}{m\mu_1}\right)^{k-m} \\ &= p_0 \left(\frac{\lambda_0}{\mu_1}\right)^k \frac{1}{m!m^{k-m}} \qquad (m \le k) \end{aligned}$$

To keep the filtering engine stable, the average filtering rate should always be higher than the average ID arrival rate (6.3)

$$\rho = \frac{\lambda_0}{m\mu_1} < 1 \tag{6.3}$$

which gives us

$$(m\rho)^k = \left(\frac{\lambda_0}{\mu_1}\right)^k, \quad \rho^k m^m = m^m \frac{{\lambda_0}^k}{m^k {\mu_1}^k} = \left(\frac{\lambda_0}{\mu_1}\right)^k \frac{1}{m^{k-m}}$$

so to summarize p_k for both cases $(0 \le k \le m)$ and $(m \le k)$:

$$p_k = \begin{cases} p_0 \dfrac{(m\rho)^k}{k!} & (0 \le k \le m) \\ p_0 \dfrac{\rho^k m^m}{m!} & (m \le k) \end{cases} \tag{6.4}$$

To reduce the number of filtering engine, we may need to know the probability (p_0) of the filtering engine to be in $i = 0$. Since we know

$$\sum_{k=0}^{\infty} p_k = 1$$

we can obtain p_0 from (6.4) by summing p over all k.

$$
\begin{aligned}
p_0 &= \left[1 + \sum_{k=1}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}\right]^{-1} \\
&= \left[1 + \sum_{k=1}^{m-1} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}} + \sum_{k=m}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}\right]^{-1} \\
&= \left[1 + \sum_{k=1}^{m-1} \frac{(m\rho)^k}{k!} + \sum_{k=m}^{\infty} \frac{\rho^k m^m}{m!}\right]^{-1} \\
&= \left[\sum_{k=0}^{1} \frac{(m\rho)^k}{k!} + \sum_{k=1}^{m-1} \frac{(m\rho)^k}{k!} + \frac{m^m}{m!} \sum_{k=m}^{\infty} \rho^k\right]^{-1} \\
&= \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{m^m}{m!} \left(\frac{\rho^m}{1-\rho}\right)\right]^{-1} \\
&= \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right)\right]^{-1}
\end{aligned}
$$

If we want the middleware to have an available filtering engine for an ID at any time, the expected queued number of ID $\overline{N}$ should also be zero ($\overline{N} = 0$ for $0 \le k \le m$).

$$
\begin{aligned}
\overline{N} &= \sum_{k=0}^{\infty} k p_k = \sum_{k=0}^{m-1} 0 \cdot p_0 \frac{(m\rho)^k}{k!} + \sum_{k=m}^{\infty} (k-m) p_0 \frac{\rho^k m^m}{m!} \\
&= \sum_{k=m}^{\infty} (k-m) \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right)\right]^{-1} \frac{\rho^k m^m}{m!} \\
&= 0
\end{aligned}
$$

In such stochastic process (a continuous Markov chain), the rate of state transition can be derived from a matrix called *Infinitesimal Generator Matrix*. (6.5) shows the infinitesimal generator matrix for M/M/m queuing system.

$$
Q = \begin{bmatrix}
-\lambda & \lambda & & & & & & & \\
\mu & -(\mu+\lambda) & \lambda & & & & & & \\
 & 2\mu & -(2\mu+\lambda) & \lambda & & & & & \\
 & & 3\mu & -(3\mu+\lambda) & \lambda & & & & \\
 & & & \ddots & \ddots & \ddots & & & \\
 & & & & c\mu & -(c\mu+\lambda) & \lambda & & \\
 & & & & & c\mu & -(c\mu+\lambda) & \lambda & \\
 & & & & & & c\mu & -(c\mu+\lambda) & \lambda \\
 & & & & & & & \ddots & \ddots
\end{bmatrix} \tag{6.5}
$$

$$
\text{when } \begin{cases} 0 \le -q_{ii} < \infty \\ 0 \le q_{ij} : \text{for } i = j \\ \sum_j q_{ij} = 0 : \text{for all } i \end{cases}
$$

With this matrix we can calculate $T/\overline{x}$ in terms of ρ to estimate the expected time T spent in the system $\overline{N}$.

6.2 RFID Event Traffic Load Balancing

LLRP Load-balancer is a supportive component to enable the parallelization in the previous section. In a form of load-balancing distributed architecture, the middleware can behave as an Entropy Filter to accommodate the dynamic incoming data by automatically scaling out for increasing tra□c or reducing the number of instances to save computing resources. As the transport for LLRP is TCP, the load balancer can be easily realized by existing solutions (e.g., Nginx[2]). The load balancer could be implemented as a hardware such as FPGA (Harik and Kayssi, 2002).

6.3 Subscription Report Aggregation

After the ID filtering process, all the resulting ID-Pattern matches are then aggregated to generate reports for the corresponding subscribers. An ALE Aggregator is the egress interface toward Event Capturing Applications and summarizes the filtered IDs passed over from multiple filtering engines. For the ingress ALE interface, Schmidt et al., 2011 proposed DHT-based distributed ALE engine for an RFID middleware, and it could be beneficial to have both of the features to realize truly scalable architecture for large amounts of RFID event subscription.

6.4 Portability and Deployment Automation

The new middleware consists of LLRP interface (with LLRP Load-Balancer), multiple Filtering Engines in parallel, and ALE Aggregator. Each component can be distributed in IP space, as they inter-communicate by an internal protocol over gRPC. They can be run as any form as a single process, Linux Containers, or VMs on Cloud. On the contrary to the recent trending in IaaS platform business model, more vendors prefers to operate this kind of middleware as On-Premises software. Infrastructure as Code and DevOps are the important factor to be included in deciding how the middleware should be installed. Al-dhuraibi et al., 2017 surveyed modern Cloud Computing Platform and its elasticity from a multifaceted point of view.

[2] https://www.nginx.com/blog/tcp-load-balancing-udp-load-balancing-nginx-tips-tricks/

Chapter 7

Conclusion

I presented an RFID middleware, enhanced by integrating Entropy Filtering mechanism to produce dynamic feedback cycles within the middleware. This design and the concept is centered in the practical discovery and the observations, so as to be applied to any future UHF-band RFID middleware implementation in any platform or hardware.

7.1 Recapitulation

There are multiple standards for coding schemes defined for IDs in RF tags vying for acceptance in industries. I first studied and clarified the binary encoding of the IDs of both GS1 and ISO standards, and make a suggestion on allocating a new namespace in URN to represent a Pure Identity for ISO IDs. Entropy Filtering enables an RFID middleware to selectively adapt the filtering engines depending on the use case and by evaluating several di□erent factors that contribute to the total entropy of the system with the dynamic feedback cycle of information. The criteria of the high performance RFID middleware are established by the preliminary experiment conducted on a open source implementation with the RFID reader emulator golemu which I developed. golemu provides a testing environment for an RFID-based system to design and verify the system integrity for the actual use. To meet the established criteria, possible four algorithms for the filtering engines; Linear Search (List), Patricia Trie, Optimal BST, and Splay Tree are studied in accordance with the parameter characteristics. Each algorithm is evaluated along with the three specific scenarios. I suggest that Patricia Trie for the terminal use cases where we have almost fixed patterns but experience high tra□c (Scenario 1), Optimal BST for the convenience store use cases where we frequently change filter patterns (Scenario 2), and List (Linear Search) for the shopping mall use cases where we filter unpredictable amount of IDs that don't match with any subscription (Scenario 3), as the preferable option. An important finding is that only by the pure throughput (e.g., ID/Sec for the filtering engine) we cannot determine the best algorithm to be used in the filtering engine, and we should take into account other criteria such as the time required for rebuilding the engine or the population of IDs to be filtered. Lastly, I concludes the thesis by making some technical suggestions on the deployment of the RFID middleware, including the parallelization of filtering engines, the method to determine an estimated performance at given state with Queuing Theory, the LLRP load balancer, and ALE Aggregator.

7.2 Future Work

The following topics are left as future works.

- Further investigation of an algorithm to handle non-prefix pattern filtering efficiently.
- Analysis on the entropy from the similarity between the subscribed filters such as the Hu□man distance
- ALE Aggregator and LLRP load-balancer implementation and the experiment between golemu and gosstrak-fc

Appendix A

Multi-code Binary Encoding Summary

This appendix shows some binary encoding examples of the multi-code IDs and the corresponding filters.

A.1 EPC

A.1.1 SGTIN-96

EPC	00110000011011011011010110110010111001010100 00000000010110000000000000000000000000000000000001 01010110
PC	3000

SGTIN-96.3	00110000011
SGTIN-96.3.3.458960468	00110000011011011011010110110010111001010100
SGTIN-96.3.3.458960468 .0022	00110000011011011011010110110010111001010100 00000000010110
SGTIN-96.3.3.458960468 .0022.86	00110000011011011011010110110010111001010100 00000000010110000000000000000000000000000000000001 01010110

A.1.2 SSCC-96

EPC	00110001011011011011010110110010111001010100 00000000000000000100110100100000000000000000 00000000
PC	3000

SSCC-96.3	00110001011
SSCC-96.3.3.458960468	00110001011011011011010110110010111001010100
SSCC-96.3.3.458960468 .1234	00110001011011011011010110110010111001010100 00000000000000000010011010010

A.1.3 GRAI-96

EPC	00110100011011011011010110110010111001010100 00000000000000000000000000000000000010110 00101110
PC	3000

GIAI-96.3	00110100011
GIAI-96.3.3.458960468	001101000110110110110101101100101110010101000
GIAI-96.3.3.458960468 .5678	00110100011011011011010110110010111001010100 00000000000000000000000000000000000010110 00101110

A.1.4 GIAI-96

EPC	00110011011011011011010110110010111001010100 00000011011100000000000000000000000000000000 00000001
PC	3000

GRAI-96.3	00110011011
GRAI-96.3.3.458960468	001100110110110110110101101100101110010101000
GRAI-96.3.3.458960468 .220	001100110110110110110101101100101110010101000 00000011011100
GRAI-96.3.3.458960468 .220.1	00110011011011011011010110110010111001010100 00000011011100000000000000000000000000000000 00000001

A.2 ISO

A.2.1 17363

UII	11011100001000000100001000001101010111000111001011001111010011010111011011000010
PC	29A9

ISO17363.7B	110111000010
ISO17363.7B.ABC	11011100001000000100001000001101010111
ISO17363.7B.ABC.U	110111000010000001000010000011010101
ISO17363.7B.ABC.U .123456	11011100001000000100001000001101010111000111001011001111010011010111011011000000

A.2.2 17365

UII	1100101101010100110101010011100000010000100000111100000101000010000000001001110001011110000011001001111010101110000000110001111010010110000010010000101000001000100001001001110000111110000010100001001000001001010011110001100000010
PC	71A2

ISO17365.25S	110010110101010011
ISO17365.25S.UN	1100101101010100110101010011 10
ISO17365.25S.UN.ABC	110010110101010011010101001110000001000010000011
ISO17365.25S.UN.ABC .0THANK0YOU0FOR 0READING0THIS1	1100101101010100110101010011100000010000100000111100000101000010000000001001110001011110000011001001111010101110000000110001111010010110000010010000101000001000100001001001110000111110000010100001001000001001010011110001

Appendix B

uiigen – A command line tool to generate EPC/ISO standard comformable IDs

B.1 Installation and Synopsis

```
% go get -v github.com/iomz/go-llrp
% uiigen --help
usage: uiigen [<flags>] <command> [<args> ...]

A tool to generate an arbitrary UII (aka EPC).

Flags:
  --help  Show context-sensitive help (also try --help-long and
 ↪  --help-man).
  --pf    Print a prefix filter for the given parameter

Commands:
  help [<command>...]
    Show help.

  epc [<flags>]
    Generate an EPC.

  iso 17363 [<flags>]
    ISO 17363 coding scheme.

  iso 17365 [<flags>]
    ISO 17365 coding scheme.

% uiigen epc --help
usage: uiigen epc [<flags>]

Generate an EPC.

Flags:
  --help        Show context-sensitive help (also try --help-long and
                --help-man).
```

```
  --pf          Print a prefix filter for the given parameter
  --cs=CS       EPC coding scheme. ex.) sgtin-96
  --filter="3"  Filter Value for EPC.
  --cp=CP       [GIAI,GRAI,SGTIN,SSCC] Company Prefix for EPC. ex.)
 ↪  0614141
  --ir=IR       [SGTIN] Item Reference Value for EPC.
  --ext=EXT     [SSCC] Extension value for EPC.
  --ser=SER     [GRAI,SGTIN] Serial value for EPC.
  --iar=IAR     [GIAI] Indivisual Asset Reference value for EPC.
  --at=AT       [GRAI] Asset Type for EPC.

% uiigen iso 17363 --help
usage: uiigen iso 17363 [<flags>]

ISO 17363 coding scheme.

Flags:
  --help     Show context-sensitive help (also try --help-long and
 ↪  --help-man).
  --pf       Print a prefix filter for the given parameter
  --oc=OC    A three letter container owner code (OC) assigned in
 ↪  cooperation
             with the Bureau International des Containers et du
 ↪  Transport
             Intermodal(BIC). ex.) CSQ
  --ei="U"   A one letter equipment category identifier (EI) in ISO
 ↪  6346 = U, J
             or Z.
  --csn=CSN  A six digit serial number (CSN). ex.) 305438

% uiigen iso 17365 --help
usage: uiigen iso 17365 [<flags>]

ISO 17365 coding scheme.

Flags:
  --help      Show context-sensitive help (also try --help-long and
 ↪  --help-man).
  --pf        Print a prefix filter for the given parameter
  --di="25S"  Data Identifier in ISO/IEC 15418. ex.) 25S
  --iac=IAC   1-3 Alphabet letters for Issuing Agency Code in ISO
 ↪  15459. ex.) UN
              or OD
  --cin=CIN   Company Identification. ex.) 043325711 or CIN1
  --sn=SN     Serial Number for ISO UII. ex.) MH8031200000000001 or
              0000000RTIA1B2C3DOSN12345

% uiigen epc --cs sgtin-96 --cp 458960468 --ir 102 --ser 1
3000,00110000011011011011010110110010111001010100000000001100...
```

Appendix C

golemu – An RFID reader emulator

C.1 Installation and Synopsis

```
% go get github.com/iomz/golemu
% golemu --help

usage: golemu [<flags>] <command> [<args> ...]

A mock LLRP-based logical reader for RFID Tags.

Flags:
      --help                   Show context-sensitive help (also try
 ▯→  --help-long and --help-man).
  -v, --verbose                Enable verbose mode.
  -m, --initialMessageID=1000  The initial messageID to start from.
  -k, --initialKeepaliveID=80000
                               The initial keepaliveID to start from.
  -p, --port=5084              LLRP listening port.
  -i, --ip=0.0.0.0             LLRP listening address.
      --version                Show application version.

Commands:
  help [<command>...]
    Show help.

  server [<flags>]
    Run as a tag stream server.

  client
    Run as a client mode.
Run as a server, listen 5084 port for LLRP incoming connection, 8080
 ▯→  port for websocket UI

% golemu server
#Access http://localhost:8080 for Web GUI
```

API SUMMARY

API METHODS - DEFAULT

addTags
deleteTags

golemu

API and SDK Documentation

Version: 1.0.0

This is an API documentation for an RFID reader emulator golemu. You can find out more about golemu at https://iomz.github.io/golemu/.

Default

addTags

Add new tags to the reader cycle

PUT

/tags

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X PUT "http://golemu.autoidlab.jp/v1/tags"
```

Parameters

Body parameters

Name	Description
body *	

Responses

Status: 405 - Invalid input

deleteTags

Delete tags from the reader cycle

DELETE

/tags

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X DELETE "http://golemu.autoidlab.jp/v1/tags"
```

Parameters

Body parameters

Name	Description
body *	

Responses

Status: 405 - Invalid input

Suggestions, contact, support and error reporting;

Contact Info: iomz@autoidlab.jp

MIT
https://opensource.org/licenses/MIT

Generated 2018-01-11T15:44:33.520Z

Appendix D

gosstrak-fc – An RFID middleware

D.1 Installation and Synopsis

```
% go get -v github.com/iomz/go-llrp
% gosstrak-fc --help
usage: gosstrak-fc [<flags>] <command> [<args> ...]

An RFID middleware to replace Fosstrak F&C.

Flags:
      --help               Show context-sensitive help (also try
 ▯→  --help-long and
                           --help-man).
  -v, --debug              Enable verbose mode.
  -f, --filter-file="filters.csv"
                           A CSV file contains filter and notify.
  -i, --id-file="ids.gob"  A gob file contains ids.
  -e, --engine-file="/var/tmp/gosstrak-fc-cache/engine.gob"
                           Indicate the filename storing the
 ▯→  filtering engine.
  -o, --out-file="/var/tmp/gosstrak-fc-cache/out.gob"
                           Indicate the filename for saving the
 ▯→  result.
  -r, --rebuild            Rebuild the filtering engine.
      --random             Randomize id order.
      --version            Show application version.

Commands:
  help [<command>...]
    Show help.

  analyze [<flags>]
    Analyze the tree node reference locality.

  dumb
    Run in dumb filter mode.
```

```
  list
    Use list of byte filtering engine.

  obst
    Use Optimal Binary Search Tree filtering engine.

  patricia
    Use Patricia Trie filtering engine.

  splay
    Use Splay Tree filtering engine.


% ls
filters.csv  ids.csv
% head
% head -n 5 ids.csv
21a2,11001011010101001101011100101000110000000100000111100011000110000
21a2,11001011010101001101011100101000110000000100000111100011001010000
21a2,11001011010101001101011100101000110000000100000111100011001110000
21a2,11001011010101001101011100101000110000000100000111100011010010000
21a2,11001011010101001101011100101000110000000100000111100011010110000
% head -n 5 filters.csv
GIAI-96_3_5_3257600,00110100011101001100011011010100000000
GIAI-96_3_5_3503215,00110100011101001101010111010001101111
GIAI-96_3_5_5760652,00110100011101010101111110011010001100
GIAI-96_3_5_6398195,00110100011101011000011010000011110011
GRAI-96_3_5_3503215,00110011011101001101010111010001101111
% gosstrak-fc patricia -r
2018/01/11 11:58:05 Loaded 63 filters from filters.csv
2018/01/11 11:58:05 Saved the Patricia Trie filtering engine to
 ↪  /var/tmp/gosstrak-fc-cache/engine.gob
2018/01/11 11:58:05 ids.gob not found, using ids.csv instead...
2018/01/11 11:58:05 Saved the result to
 ↪  /var/tmp/gosstrak-fc-cache/out.gob
% ls
filters.csv  ids.csv      ids.gob
% ls -al /var/tmp/gosstrak-fc-cache/
total 264
drwxr-xr-x  4 iomz  wheel     128 Dec 17 15:13 ./
drwxrwxrwt  6 root  wheel     192 Jan 10 16:09 ../
-rw-r--r--  1 iomz  wheel   23595 Jan 11 11:58 engine.gob
-rw-r--r--  1 iomz  wheel  106927 Jan 11 11:58 out.gob
```

D.2 Implementation of the Selected Algorithms

This section shows sample implementations in Go for the algorithms explained in chapter 4.

D.2.1 List (Linear Search)

The following is a snippet of source code for the List filtering engine of golemu as defined in 4.

```
// Copyright (c) 2018 Iori Mizutani

package filtering

import (
        "bytes"
        "encoding/gob"
        "errors"
        "fmt"
)

// List is a slice of pointers to ExactMatch
type List []*ExactMatch

// ExactMatch is a raw filter directly taken from Subscriptions
type ExactMatch struct {
        notificationURI string
        filter          *FilterObject
}

// MarshalBinary overwrites the marshaller in gob encoding *List
func (list *List) MarshalBinary() (_ []byte, err error) {
        var buf bytes.Buffer
        enc := gob.NewEncoder(&buf)

        // Type of Engine
        enc.Encode("Engine:filtering.List")

        // Size of List
        enc.Encode(len(*list))
        for _, em := range *list {
                // Notify
                enc.Encode(em.notificationURI)
                // Filter
                err = enc.Encode(em.filter)
        }

        return buf.Bytes(), err
}

// Search returns a slice of notificationURI
func (list *List) Search(id []byte) (matches []string) {
        for _, em := range *list {
                if em.filter.Match(id) {
                        matches = append(matches, em.notificationURI)
                }
        }
```

```
48          return
49  }
50
51  // UnmarshalBinary overwrites the unmarshaller in gob decoding List
52  func (list *List) UnmarshalBinary(data []byte) (err error) {
53          dec := gob.NewDecoder(bytes.NewReader(data))
54
55          // Type of Engine
56          var typeOfEngine string
57          if err = dec.Decode(&typeOfEngine); err != nil ||
            ▯→  typeOfEngine != "Engine:filtering.List" {
58                  return errors.New("Wrong Filtering Engine: " +
                    ▯→  typeOfEngine)
59          }
60
61          // Size of List
62          var listSize int
63          if err = dec.Decode(&listSize); err != nil {
64                  return
65          }
66
67          for i := 0; i < listSize; i++ {
68                  em := ExactMatch{}
69                  // Notify
70                  if err = dec.Decode(&em.notificationURI); err != nil
                    ▯→  {
71                          return
72                  }
73                  // Filter
74                  err = dec.Decode(&em.filter)
75                  *list = append(*list, &em)
76          }
77
78          return
79  }
80
81  // BuildList builds a simple list of filters from
    ▯→  filter.Subscriptions
82  // returns the pointer to the slice of ExactMatch struct
83  func BuildList(sub Subscriptions) *List {
84          list := make(List, len(sub))
85
86          // store ExactMatch in sorted order from sub
87          for i, fs := range sub.keys() {
88                  list[i] = &ExactMatch{sub[fs].NotificationURI,
                    ▯→  NewFilter(fs, 0)}
89          }
90
91          return &list
92  }
```

D.2.2 Patricia Trie

The following is a snippet of source code for the Patricia Trie filtering engine of golemu as defined in 4.

```
1  // Copyright (c) 2018 Iori Mizutani
2
3  package filtering
4
5  import (
6          "bytes"
7          "encoding/gob"
8          "errors"
9          "fmt"
10         "io"
11         "reflect"
12         "strings"
13 )
14
15 // PatriciaTrie struct
16 type PatriciaTrie struct {
17         notificationURI string
18         filterObject    *FilterObject
19         one             *PatriciaTrie
20         zero            *PatriciaTrie
21 }
22
23 // MarshalBinary overwrites the marshaller in gob encoding
   ↪  *PatriciaTrie
24 func (pt *PatriciaTrie) MarshalBinary() (_ []byte, err error) {
25         var buf bytes.Buffer
26         enc := gob.NewEncoder(&buf)
27
28         // Type of Engine
29         enc.Encode("Engine:filtering.PatriciaTrie")
30
31         // Notify
32         enc.Encode(pt.notificationURI)
33
34         // Filter
35         hasFilter := pt.filterObject != nil
36         enc.Encode(hasFilter)
37         if hasFilter {
38                 err = enc.Encode(pt.filterObject)
39         }
40
41         // One
42         hasOne := pt.one != nil
43         enc.Encode(hasOne)
44         if hasOne {
45                 err = enc.Encode(pt.one)
46         }
```

```
47
48          // Zero
49          hasZero := pt.zero != nil
50          enc.Encode(hasZero)
51          if hasZero {
52                  err = enc.Encode(pt.zero)
53          }
54
55          //buf.Encode
56          return buf.Bytes(), err
57  }
58
59  // Search returns a slice of notificationURI
60  func (pt *PatriciaTrie) Search(id []byte) (matches []string) {
61          // if not match, return empty slice immediately
62          if !pt.filterObject.Match(id) {
63                  return
64          }
65
66          // if the id matched with this node, return notificationURI
67          if len(pt.notificationURI) != 0 {
68                  matches = append(matches, pt.notificationURI)
69          }
70
71          // Determine next filter
72          nextBitOffset := pt.filterObject.Offset +
            ▯→  pt.filterObject.Size
73          nb, err := getNextBit(id, nextBitOffset)
74          if err != nil {
75                  panic(err)
76          }
77          if nb == '1' && pt.one != nil {
78                  matches = append(matches, pt.one.Search(id)...)
79          } else if nb == '0' && pt.zero != nil {
80                  matches = append(matches, pt.zero.Search(id)...)
81          }
82          return
83  }
84
85  // UnmarshalBinary overwrites the unmarshaller in gob decoding
    ▯→  *PatriciaTrie
86  func (pt *PatriciaTrie) UnmarshalBinary(data []byte) (err error) {
87          dec := gob.NewDecoder(bytes.NewReader(data))
88
89          // Type of Engine
90          var typeOfEngine string
91          if err = dec.Decode(&typeOfEngine); err != nil ||
            ▯→  typeOfEngine != "Engine:filtering.PatriciaTrie" {
92                  return errors.New("Wrong Filtering Engine: " +
                    ▯→  typeOfEngine)
93          }
```

```
94
95          // Notify
96          if err = dec.Decode(&pt.notificationURI); err != nil {
97                  return
98          }
99
100         // Filter
101         var hasFilter bool
102         if err = dec.Decode(&hasFilter); err != nil {
103                 return
104         }
105         if hasFilter {
106                 err = dec.Decode(&pt.filterObject)
107         } else {
108                 pt.filterObject = nil
109         }
110
111         // One
112         var hasOne bool
113         if err = dec.Decode(&hasOne); err != nil {
114                 return
115         }
116         if hasOne {
117                 err = dec.Decode(&pt.one)
118         } else {
119                 pt.one = nil
120         }
121
122         // Zero
123         var hasZero bool
124         if err = dec.Decode(&hasZero); err != nil {
125                 return
126         }
127         if hasOne {
128                 err = dec.Decode(&pt.zero)
129         } else {
130                 pt.zero = nil
131         }
132
133         return
134 }
135
136 // Internal helper methods
    ▯→  ------------------------------------------------------
137
138 func (pt *PatriciaTrie) build(prefix string, sub Subscriptions) {
139         onePrefixBranch := ""
140         zeroPrefixBranch := ""
141         fks := sub.keys()
142         for _, fk := range fks {
143                 // if the prefix is already longer than the testee
```

```
144                 if len(fk) < len(prefix) {
145                         continue
146                 }
147                 // ignore the testee without the prefix
148                 if !strings.HasPrefix(fk, prefix) {
149                         continue
150                 }
151                 p := fk[len(prefix):]
152                 // ignore if no remainder
153                 if len(p) == 0 {
154                         continue
155                 }
156                 // if the remainder starts with 1
157                 if strings.HasPrefix(p, "1") {
158                         if len(onePrefixBranch) == 0 {
159                                 onePrefixBranch = p
160                         } else {
161                                 onePrefixBranch = lcp([]string{p,
                                    ↪  onePrefixBranch})
162                         }
163                         // if the remainder starts with 0
164                 } else if strings.HasPrefix(p, "0") {
165                         if len(zeroPrefixBranch) == 0 {
166                                 zeroPrefixBranch = p
167                         } else {
168                                 zeroPrefixBranch = lcp([]string{p,
                                    ↪  zeroPrefixBranch})
169                         }
170                 }
171         }
172         cumulativePrefix := ""
173         // if there's a branch starts with 1
174         if len(onePrefixBranch) != 0 {
175                 pt.one = &PatriciaTrie{}
176                 pt.one.filterObject = NewFilter(onePrefixBranch,
                    ↪  len(prefix))
177                 cumulativePrefix = prefix + onePrefixBranch
178                 // check if the prefix matches whole filter
179                 if info, ok := sub[cumulativePrefix]; ok {
180                         pt.one.notificationURI = info.NotificationURI
181                 }
182                 pt.one.build(cumulativePrefix, sub)
183         }
184         // if there's a branch starts with 0
185         if len(zeroPrefixBranch) != 0 {
186                 pt.zero = &PatriciaTrie{}
187                 pt.zero.filterObject = NewFilter(zeroPrefixBranch,
                    ↪  len(prefix))
188                 cumulativePrefix = prefix + zeroPrefixBranch
189                 // check if the prefix matches whole filter
190                 if info, ok := sub[cumulativePrefix]; ok {
```

```
                        pt.zero.notificationURI =
                        ▯→  info.NotificationURI
                }
                pt.zero.build(cumulativePrefix, sub)
        }
}

// BuildPatriciaTrie builds PatriciaTrie from filter.Subscriptions
// returns the pointer to the node node
func BuildPatriciaTrie(sub Subscriptions) *PatriciaTrie {
        p1 := lcp(sub.keys())
        if len(p1) == 0 {
                // do something if there's no common prefix
        }
        head := &PatriciaTrie{}
        head.filterObject = NewFilter(p1, 0)
        head.build(p1, sub)

        return head
}

func getNextBit(id []byte, nbo int) (rune, error) {
        o := nbo / ByteLength
        // No more bit in the ID
        if len(id) == o && nbo%ByteLength == 0 {
                return 'x', nil
        }
        if len(id) <= o {
                return '?', errors.New("getNextBit error")
        }
        if (uint8(id[o])>>uint8((7-(nbo%ByteLength))))%2 == 0 {
                return '0', nil
        }
        return '1', nil
}

// lcp finds the longest common prefix of the input strings.
// It compares by bytes instead of runes (Unicode code points).
// It's up to the caller to do Unicode normalization if desired
// (e.g. see golang.org/x/text/unicode/norm).
func lcp(l []string) string {
        // Special cases first
        switch len(l) {
        case 0:
                return ""
        case 1:
                return l[0]
        }
        // LCP of min and max (lexigraphically)
        // is the LCP of the whole set.
        min, max := l[0], l[0]
```

```
241         for _, s := range l[1:] {
242                 switch {
243                 case s < min:
244                         min = s
245                 case s > max:
246                         max = s
247                 }
248         }
249         for i := 0; i < len(min) && i < len(max); i++ {
250                 if min[i] != max[i] {
251                         return min[:i]
252                 }
253         }
254         // In the case where lengths are not equal but all bytes
255         // are equal, min is the answer ("foo" < "foobar").
256         return min
257 }
```

D.2.3 Optimal Binary Search Tree (Optimal BST)

The following is a snippet of source code for the Optimal BST filtering engine of golemu as defined in 4.

```
1  // Copyright (c) 2018 Iori Mizutani
2
3  package filtering
4
5  import (
6          "bytes"
7          "encoding/gob"
8          "errors"
9          "fmt"
10         "io"
11         "reflect"
12         "strings"
13 )
14
15 // OptimalBST struct
16 type OptimalBST struct {
17         notificationURI string
18         filterObject    *FilterObject
19         matchNext       *OptimalBST
20         mismatchNext    *OptimalBST
21 }
22
23 // MarshalBinary overwrites the marshaller in gob encoding
   ↪  *OptimalBST
24 func (obst *OptimalBST) MarshalBinary() (_ []byte, err error) {
25         var buf bytes.Buffer
26         enc := gob.NewEncoder(&buf)
27
```

```
28          // Type of Engine
29          enc.Encode("Engine:filtering.OptimalBST")
30
31          // Notify
32          enc.Encode(obst.notificationURI)
33
34          // Filter
35          hasFilter := obst.filterObject != nil
36          enc.Encode(hasFilter)
37          if hasFilter {
38                  err = enc.Encode(obst.filterObject)
39          }
40
41          // matchNext
42          hasMatchNext := obst.matchNext != nil
43          enc.Encode(hasMatchNext)
44          if hasMatchNext {
45                  err = enc.Encode(obst.matchNext)
46          }
47
48          // mismatchNext
49          hasMismatchNext := obst.mismatchNext != nil
50          enc.Encode(hasMismatchNext)
51          if hasMismatchNext {
52                  err = enc.Encode(obst.mismatchNext)
53          }
54
55          return buf.Bytes(), err
56  }
57
58  // Search returns a slice of notificationURI
59  func (obst *OptimalBST) Search(id []byte) []string {
60          matches := []string{}
61          if obst.filterObject.Match(id) {
62                  matches = append(matches, obst.notificationURI)
63                  if obst.matchNext != nil {
64                          matches = append(matches,
                            ▯→  obst.matchNext.Search(id)...)
65                  }
66                  return matches
67          }
68          if obst.mismatchNext != nil {
69                  return obst.mismatchNext.Search(id)
70          }
71          return matches
72  }
73
74  // UnmarshalBinary overwrites the unmarshaller in gob decoding
    ▯→  *PatriciaTrie
75  func (obst *OptimalBST) UnmarshalBinary(data []byte) (err error) {
76          dec := gob.NewDecoder(bytes.NewReader(data))
```

```
77
78          // Type of Engine
79          var typeOfEngine string
80          if err = dec.Decode(&typeOfEngine); err != nil ||
            ↪  typeOfEngine != "Engine:filtering.OptimalBST" {
81                  return errors.New("Wrong Filtering Engine: " +
                    ↪  typeOfEngine)
82          }
83
84          // Notify
85          if err = dec.Decode(&obst.notificationURI); err != nil {
86                  return
87          }
88
89          // FilterObject
90          var hasFilterObject bool
91          if err = dec.Decode(&hasFilterObject); err != nil {
92                  return
93          }
94          if hasFilterObject {
95                  err = dec.Decode(&obst.filterObject)
96          }
97
98          // matchNext
99          var hasMatchNext bool
100         if err = dec.Decode(&hasMatchNext); err != nil {
101                 return
102         }
103         if hasMatchNext {
104                 err = dec.Decode(&obst.matchNext)
105         } else {
106                 obst.matchNext = nil
107         }
108
109         // mismatchNext
110         var hasMismatchNext bool
111         if err = dec.Decode(&hasMismatchNext); err != nil {
112                 return
113         }
114         if hasMismatchNext {
115                 err = dec.Decode(&obst.mismatchNext)
116         } else {
117                 obst.mismatchNext = nil
118         }
119
120         return
121 }
122
123 // Internal helper methods
    ↪  ----------------------------------------------------
124
```

```
125 func (obst *OptimalBST) build(sub *Subscriptions, nds *Nodes)
    ▯→  *OptimalBST {
126         current := obst
127         for i, nd := range *nds {
128                 current.filterObject = NewFilter(nd.filter,
                    ▯→  nd.offset)
129                 current.notificationURI = nd.notificationURI
130                 // if this node has subset
131                 if _, ok := (*sub)[nd.filter]; ok &&
                    ▯→  (*sub)[nd.filter].Subset != nil {
132                         subset := (*sub)[nd.filter].Subset
133                         subnds := NewNodes(subset)
134                         subobst := &OptimalBST{}
135                         current.matchNext = subobst.build(subset,
                            ▯→  subnds)
136                 } else {
137                         current.matchNext = nil
138                 }
139                 if i+1 < len(*nds) {
140                         current.mismatchNext = &OptimalBST{}
141                         current = current.mismatchNext
142                 } else {
143                         current.mismatchNext = nil
144                 }
145         }
146         return obst
147 }
148
149 // BuildOptimalBST builds OptimalBST from Subscriptions
150 // returns the pointer to the node node
151 func BuildOptimalBST(sub *Subscriptions) *OptimalBST {
152         // make subsets to the child subscriptions of the
            ▯→  corresponding parents
153         sub.linkSubset()
154
155         // recalculate the cumulative evs if it has subset
            ▯→  subscriptions
156         for _, info := range *sub {
157                 if info.Subset != nil {
158                         info.EntropyValue +=
                            ▯→  recalculateEntropyValue(info.Subset)
159                 }
160         }
161
162         nds := NewNodes(sub)
163         obst := &OptimalBST{}
164         return obst.build(sub, nds)
165 }
```

D.2.4 Splay Tree

The following is a snippet of source code for the Splay Tree filtering engine of golemu as defined in 4.

```
1  // Copyright (c) 2018 Iori Mizutani
2
3  package filtering
4
5  import (
6          "bytes"
7          "encoding/gob"
8          "errors"
9  )
10
11 // SplayTree struct
12 type SplayTree struct {
13         root *OptimalBST
14 }
15
16 // MarshalBinary overwrites the marshaller in gob encoding
   ▯→  *SplayTree
17 func (spt *SplayTree) MarshalBinary() (_ []byte, err error) {
18         var buf bytes.Buffer
19         enc := gob.NewEncoder(&buf)
20
21         // Type of Engine
22         enc.Encode("Engine:filtering.SplayTree")
23
24         // Encode OptimalBST
25         enc.Encode(spt.root)
26
27         return buf.Bytes(), err
28 }
29
30 // Search returns a slice of notificationURI
31 func (spt *SplayTree) Search(id []byte) []string {
32         return spt.root.splaySearch(spt, nil, id)
33 }
34
35 // UnmarshalBinary overwrites the unmarshaller in gob decoding
   ▯→  *PatriciaTrie
36 func (spt *SplayTree) UnmarshalBinary(data []byte) (err error) {
37         dec := gob.NewDecoder(bytes.NewReader(data))
38
39         // Type of Engine
40         var typeOfEngine string
41         if err = dec.Decode(&typeOfEngine); err != nil ||
           ▯→  typeOfEngine != "Engine:filtering.SplayTree" {
42                 return errors.New("Wrong Filtering Engine: " +
                   ▯→  typeOfEngine)
43         }
```

```
44
45          // Decode OptimalBST
46          err = dec.Decode(&spt.root)
47
48          return
49  }
50
51  // Internal helper methods
    ▯→  ----------------------------------------------------
52
53  func (obst *OptimalBST) splaySearch(spt *SplayTree, parent
    ▯→  *OptimalBST, id []byte) []string {
54          matches := []string{}
55          if obst.filterObject.Match(id) {
56                  matches = append(matches, obst.notificationURI)
57                  if obst.matchNext != nil {
58                          // Do not splay the subsets
59                          matches = append(matches,
                            ▯→  obst.matchNext.Search(id)...)
60                  }
61                  // Splay
62                  // 0. Check if this is the root node, do nothing if
                    ▯→  so
63                  if parent != nil {
64                          // 1. Remove this node and connect parent to
                            ▯→  the next node
65                          parent.mismatchNext = obst.mismatchNext
66                          // 2. Insert self to root
67                          obst.mismatchNext = spt.root
68                          spt.root = obst
69                  }
70                  return matches
71          }
72          if obst.mismatchNext != nil {
73                  return obst.mismatchNext.splaySearch(spt, obst, id)
74          }
75          return matches
76  }
77
78  // BuildSplayTree builds SplayTree from Subscriptions
79  // returns the pointer to the node node
80  func BuildSplayTree(sub *Subscriptions) *SplayTree {
81          // make subsets to the child subscriptions of the
            ▯→  corresponding parents
82          sub.linkSubset()
83
84          nds := NewNodes(sub)
85          spt := &SplayTree{}
86          spt.root = &OptimalBST{}
87          spt.root = spt.root.build(sub, nds)
88          return spt
```

```
}
```

D.2.5 Filter

The following is a source code for the FilterObject type in golemu to execute the byte-wise filtering mentioned in chapter 4.

```
// Copyright (c) 2018 Iori Mizutani

package filtering

import (
        "errors"
        "fmt"
        "reflect"
        "strings"

        "github.com/iomz/go-llrp/binutil"
)

const (
        // ByteLength defines a bit length of one byte
        ByteLength = 8
)

// FilterObject type is a struct for filter element
type FilterObject struct {
        String     string
        Size       int
        Offset     int
        ByteFilter []byte
        ByteMask   []byte
        ByteOffset int
        ByteSize   int
}

// GetByteAt returns a byte of ByteFilter and ByteMask
// at the given offset, returns error if HasByteAt(bo) is false
func (f *FilterObject) GetByteAt(bo int) (byte, byte, error) {
        if f.HasByteAt(bo) {
                for i := 0; i < f.ByteSize; i++ {
                        if bo == f.ByteOffset+i {
                                return f.ByteFilter[i],
                                ▯→  f.ByteMask[i], nil
                        }
                }
        }
        return 0, 0, errors.New("this filter doesn't have the byte in
        ▯→  the given offset")
}

```

```
43  // HasByteAt returns true if the ByteFilter covers
44  // a byte starting with the given offset
45  func (f *FilterObject) HasByteAt(bo int) bool {
46          if f.ByteOffset > bo { // the filter is after the given bo
47                  return false
48          } else if f.ByteOffset+f.ByteSize <= bo { // the filter is
            ↪  before the given offset
49                  return false
50          }
51          return true
52  }
53
54  // Match returns true if the id is captured by this filter
55  func (f *FilterObject) Match(id []byte) bool {
56          for i := 0; i < f.ByteSize; i++ {
57                  if (id[f.ByteOffset+i]|f.ByteMask[i])^f.ByteFilter[i]
                    ↪  != byte(0) {
58                          return false
59                  }
60          }
61          return true
62  }
63
64  // ToString returns a string representation of FilterObject
65  func (f *FilterObject) ToString() string {
66          return fmt.Sprintf("%s(%d %d)", f.String, f.Offset, f.Size)
67  }
68
69  // IsTransparent returns true if the filter is meaningless
70  // = any id will match with this filter
71  func (f *FilterObject) IsTransparent() bool {
72          check := make([]byte, f.ByteSize)
73          for i := 0; i < f.ByteSize; i++ {
74                  check[i] = 255
75          }
76          if reflect.DeepEqual(f.ByteFilter, check) &&
77                  reflect.DeepEqual(f.ByteMask, check) {
78                  return true
79          }
80          return false
81  }
82
83  // makeFilter returns padded offset, filter and mask in rune slices
84  func makeFilter(bs []rune, offset int) (int, []rune, []rune) {
85          var f []rune
86          var m []rune
87          var bodyLength int
88
89          leftPaddingLength := offset % ByteLength
90          // pad left with 1 if necessary
91          f = []rune(strings.Repeat("1", leftPaddingLength))
```

```
92          m = []rune(strings.Repeat("1", leftPaddingLength))
93
94          // insert filter body
95          if ByteLength-leftPaddingLength < len(bs) {
96                  bodyLength = ByteLength - leftPaddingLength
97          } else {
98                  bodyLength = len(bs)
99          }
100         f = append(f, bs[0:bodyLength]...)
101         m = append(m, []rune(strings.Repeat("0", bodyLength))...)
102
103         // pad right with 1 if necessary
104         if len(f) < ByteLength {
105                 rightPaddingLength := ByteLength - len(f)
106                 f = append(f, []rune(strings.Repeat("1",
                    ▯→  rightPaddingLength))...)
107                 m = append(m, []rune(strings.Repeat("1",
                    ▯→  rightPaddingLength))...)
108         }
109
110         // Apply wildcard x bits in the filter to mask
111         for i := range f {
112                 if f[i] == 'x' {
113                         f[i] = '1'
114                         m[i] = '1'
115                 }
116         }
117
118         // if there is remainder, continue making the filter
119         if len(bs)+leftPaddingLength > ByteLength {
120                 nextOffset := ((offset / ByteLength) + 1) *
                    ▯→  ByteLength
121                 _, fc, mc := makeFilter(bs[bodyLength:], nextOffset)
122                 f = append(f, fc...)
123                 m = append(m, mc...)
124         }
125
126         return offset / ByteLength, f, m
127 }
128
129 // NewFilter constructs FilterObject
130 func NewFilter(s string, o int) *FilterObject {
131         bo, f, m := makeFilter([]rune(s), o)
132         bf, _ := binutil.ParseBinRuneSliceToUint8Slice(f)
133         bm, _ := binutil.ParseBinRuneSliceToUint8Slice(m)
134
135         return &FilterObject{
136                 String:     s,
137                 Size:       len(s),
138                 Offset:     o,
139                 ByteFilter: bf,
```

```
140                     ByteMask:   bm,
141                     ByteOffset: bo,
142                     ByteSize:   len(bf),
143             }
144     }
```

Bibliography

Al-dhuraibi, Yahya et al. (2017). “Elasticity in Cloud Computing : State of the Art and Research Challenges”. In: 1374.c, pp. 1–18. ISSN: 1939-1374. DOI: 10.1109/TSC.2017.2711009.

Alien Technology, LLC. (2018). *Alien Technology Readers - Industry Leader in Ease of Use | Alien Technology*. URL: http://www.alientechnology.com/products/readers/.

Birman, K. and T. Joseph (1987). “Exploiting virtual synchrony in distributed systems”. In: *ACM SIGOPS Operating Systems Review* 21.5, pp. 123–138. ISSN: 01635980. DOI: 10.1145/37499.37515.

Blochwitz, Christopher et al. (2016). “An optimized radix-tree for hardware-accelerated dictionary generation for semantic web databases”. In: *2015 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015*. DOI: 10.1109/ReConFig.2015.7393291.

Burgard, Wolfram, Dieter Fox, and Sebastian Thrun (1997). “Active Mobile Robot Localization by Entropy Minimization”. In: *Proc.˜of the 2nd Euromicro Workshop on Advanced Mobile Robots*, pp. 1162–1505.

Catalyst Direct (2017). *EAS security tags and ink security tags*. URL: https://www.catalyst-direct.com/uk/products/eas/security-tags/.

Checkpoint Systems, Inc. (2016). *Electronic Article Surveillance (EAS)*. URL: https://us.checkpointsystems.com/solutions/apparel-labeling-solutions/products/rfid-eas/electronic-article-surveillance-eas/.

Computer Economics, Inc. (2007). *RFID Adoption Stalls: Executive Summary*. URL: https://www.computereconomics.com/article.cfm?id=1203.

Dai Nippon Printing Co., Ltd. (2017). *DNP Embarks Upon Development of Low Cost RFID Tag for Convenience Stores*. URL: https://www.dnp.co.jp/eng/news/10140395_2501.html.

Daigle, L. et al. (Oct. 2002). *Uniform Resource Names (URN) Namespace Definition Mechanisms*. BCP 66. RFC Editor.

ECMA (June 2013). *ECMA-340 Near Field Communication Interface and Protocol (NFCIP-1)*. Tech. rep.

FUJITSU FRONTECH LIMITED (2017). *Fujitsu announces partnership with Positek RFID L.P. to grow the RFID market in North America*. URL: https://www.fujitsu.com/jp/group/frontech/en/resources/news/press-releases/2017/1106.html.

– (2018). *Products : FUJITSU FRONTECH*. URL: http://www.fujitsu.com/jp/group/frontech/en/products/.

Gaudin, Sharon (2008). *Some Suppliers Gain from Failed Wal-Mart RFID Edict*. URL: https://www.cio.com/article/2436434/rfid/some-suppliers-gain-from-failed-wal-mart-rfid-edict.html.

Goodwin, J. and H. Apel (Mar. 2008). *A Uniform Resource Name (URN) Namespace for the International Organization for Standardization (ISO)*. RFC 5141. RFC Editor.

GS1 (Mar. 2009). *Application Level Event (ALE) Standard*. Tech. rep. URL: https://www.gs1.org/ale.
– (Oct. 2010). *Low Level Reader Protocol*. Tech. rep. URL: https://www.gs1.org/epcrfid/epc-rfid-llrp/1-1-0.
– (Oct. 2011). *Tag Data Translation Standard*. Tech. rep. URL: https://www.gs1.org/epcrfid-epcis-id-keys/epc-rfid-tdt/1-6.
– (Apr. 2014). *EPCglobal Architecture Framework*. Tech. rep. URL: https://www.gs1.org/id-keys-epcrfid-epcis/epc-rfid-architecture-framework/1-6.
– (Apr. 2015). *UHF Air Interface Protocol Standard*. Tech. rep. URL: https://www.gs1.org/epcrfid/epc-rfid-uhf-air-interface-protocol/2-0-1.
– (2016). *GS1 RFID/Barcode Interoperability Guideline*. URL: https://www.gs1.org/docs/barcodes/RFID_Barcode_Interoperability_Guidelines.pdf.
– (Sept. 2017a). *EPC Tag Data Standard (TDS)*. Tech. rep. URL: https://www.gs1.org/epcrfid-epcis-id-keys/epc-rfid-tds/1-11.
– (July 2017b). *GS1 General Specifications*. Tech. rep. URL: https://www.gs1.org/barcodes-epcrfid-id-keys/gs1-general-specifications.
Gupta, Pankaj (2000). "Algorithms for routing lookups and packet classification". In: *Computer Science* PhD.December.
Harik, L. and A. Kayssi (2002). "FPGA-based load balancer for Internet servers". In: *Proceedings of the International Conference on Microelectronics, ICM*. Vol. 2002-January, pp. 190–193. ISBN: 0780375734. DOI: 10.1109/ICM-02.2002.1161527.
Hermann, Mario, Tobias Pentek, and Boris Otto (2016). "Design principles for industrie 4.0 scenarios". In: *Proceedings of the Annual Hawaii International Conference on System Sciences*. Vol. 2016-March, pp. 3928–3937. ISBN: 9780769556703. DOI: 10.1109/HICSS.2016.488. arXiv: arXiv:1011.1669v3.
Honeywell International Inc. (2018). *Honeywell RFID*. URL: https://www.honeywellaidc.com/products/rfid.
Howes, Tim et al. (Mar. 1995). *The String Representation of Standard Attribute Syntaxes*. RFC 1778. https://www.rfc-editor.org/rfc/rfc1778.txt. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc1778.txt.
Huebner, A., C. Facchi, and H. Janicke (2012). "Rifidi Toolkit : Virtuality for Testing RFID Systems". In: *ICSNC 2012*, pp. 1–6.
Impinj, Inc. (2018). *RAIN RFID Reader, Gateway & Chip Selector, Connectivity | Impinj*. URL: https://www.impinj.com/platform/connectivity/.
International Air Transport Association (IATA) (2017). *Guidelines - Introducing RFID into Airline Maintenance Operations*. URL: https://www.iata.org/publications/Pages/rfid-maintenance-ops.aspx.
ISO (Nov. 1995). *ISO 6346:1995 Freight containers – Coding, identification and marking*. Tech. rep.
– (Apr. 2011). *ISO 14223-1:2011 Radiofrequency identification of animals – Advanced transponders – Part 1: Air interface*. Tech. rep.
– (Mar. 2013a). *ISO 17363:2013 Supply chain applications of RFID – Freight containers*. Tech. rep.
– (Feb. 2013b). *ISO 17364:2013 Supply chain applications of RFID – Returnable transport items (RTIs) and returnable packaging items (RPIs)*. Tech. rep.
– (Feb. 2013c). *ISO 17365:2013 Supply chain applications of RFID – Transport units*. Tech. rep.
– (Feb. 2013d). *ISO 17366:2013 Supply chain applications of RFID – Product packaging*. Tech. rep.
– (Feb. 2013e). *ISO 17367:2013 Supply chain applications of RFID – Product tagging*. Tech. rep.

ISO/IEC (Mar. 2006). *ISO/IEC 15434:2006 Information technology – Automatic identification and data capture techniques – Syntax for high-capacity ADC media*. Tech. rep.
– (Aug. 2009a). *ISO/IEC 15418:2009 Information technology - Automatic identification and data capture techniques - GS1 Application Identifiers and ASC MH10 Data Identifiers and maintenance*. Tech. rep.
– (Oct. 2009b). *ISO/IEC 18000-2:2009 Information technology – Radio frequency identification for item management – Part 2: Parameters for air interface communications below 135 kHz*. Tech. rep.
– (Oct. 2010a). *ISO/IEC 15693-1:2010 Identification cards – Contactless integrated circuit cards – Vicinity cards – Part 1: Physical characteristics*. Tech. rep.
– (Apr. 2010b). *ISO/IEC15961-1:2010 Information technology – Radio frequency identification (RFID) for item management: Data protocol – Part 1: Application interface*. Tech. rep.
– (Oct. 2011). *ISO/IEC 24791-2:2011 Information technology – Radio frequency identification (RFID) for item management – Software system infrastructure – Part 2: Data management*. Tech. rep.
– (Dec. 2012). *ISO/IEC 24791-5:2012 Information technology – Radio frequency identification (RFID) for item management – Software system infrastructure – Part 5: Device interface*. Tech. rep.
– (Mar. 2013a). *ISO/IEC 15962:2013 Information technology – Radio frequency identification (RFID) for item management – Data protocol: data encoding rules and logical memory functions*. Tech. rep.
– (Feb. 2013b). *ISO/IEC 18000-63:2013 Information technology – Radio frequency identification for item management – Part 63: Parameters for air interface communications at 860 MHz to 960 MHz Type C*. Tech. rep.
– (Mar. 2013c). *ISO/IEC 18092:2013 Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)*. Tech. rep.
– (Nov. 2014a). *ISO/IEC 15459-1:2014 Information technology – Automatic identification and data capture techniques – Unique identification – Part 1: Individual transport units*. Tech. rep.
– (Nov. 2014b). *ISO/IEC15459-4:2014 Information technology – Automatic identification and data capture techniques – Unique identification – Part 4: Individual products and product packages*. Tech. rep.
– (Nov. 2014c). *ISO/IEC15459-5:2014 Information technology – Automatic identification and data capture techniques – Unique identification – Part 5: Individual returnable transport items (RTIs)*. Tech. rep.
– (Nov. 2014d). *ISO/IEC15459-6:2014 Information technology – Automatic identification and data capture techniques – Unique identification – Part 6: Groupings*. Tech. rep.
– (Feb. 2015). *ISO/IEC 18000-4:2015 Information technology – Radio frequency identification for item management – Part 4: Parameters for air interface communications at 2,45 GHz*. Tech. rep.
– (Mar. 2016). *ISO/IEC 14443-1:2016 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 1: Physical characteristics*. Tech. rep.
ISO/IEC FDIS (Apr. 2010). *ISO/IEC FDIS 15961-2:2010 Information technology – Radio frequency identification (RFID) for item management: Data protocol – Part 2: Registration of RFID data constructs*. Tech. rep.

Japan Institute of Logistics Systems (2013). *Container Round-use and Ecological Logistics, Survey in 2013 (Japanese)*. URL: https://www.logistics.or.jp/jils_news/2013fy_survey2_container_round.pdf.

JIS (July 2005). *JIS X 6319-4 Specification of Implementation for integrated circuit(s) cards Part 4: High Speed proximity cards*. Tech. rep.

Kleinrock, Leonard (1975). *QUEUEING SYSTEMS Volume I: Theory*. John Wiley & Sons, Inc.

Knuth, Donald (1971). "Optimum Binary Search Trees". In: *Acta Informatica* 1, pp. 14–25.

Mealling, M. (Nov. 2000). *A URN Namespace of Object Identifiers*. RFC 3001. RFC Editor.

– (Feb. 2001). *A URN Namespace of Object Identifiers*. RFC 3061. RFC Editor.

– (Jan. 2008). *A Uniform Resource Name Namespace for the EPCglobal Electronic Product Code (EPC) and Related Standards*. RFC 5134. RFC Editor.

Mighty Cube Co.,Ltd. (2017). *Anti-Theft system (EAS & DMP)*. URL: https://www.mightycube.co.jp/en/solution/anti-theft/.

Mitsubishi Electric Corporation (2018). *UHF RFID Reader | Mitsubishi Electric Corporation*. URL: https://www.mitsubishielectric.co.jp/device/rfid/products/index.html.

Moats, R. (May 1997). *URN Syntax*. RFC 2141. RFC Editor.

Morrison, Donald R. (1968). "PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric". In: *Journal of the ACM* 15.4, pp. 514–534. ISSN: 00045411. DOI: 10.1145/321479.321481. URL: https://portal.acm.org/citation.cfm?doid=321479.321481.

NEC Corporation (2015). *RFID Middleware Products : WebOTX | NEC*. URL: https://www.nec.com/en/global/prod/webotx/en/rfid.html.

NTT Electronics (2014). *First Realization of "Smart Pallet System" with 920 Mhz Active RFID with Cloud (Japanese)*. URL: https://www.ntt-electronics.com/new/information/2014/9/upr-rfid-smart-pallet.html.

OMRON Corporation (2018). *Vision & Identification systems | Omron, United States*. URL: https://industrial.omron.us/en/products/-inspection-ident-systems.

Oracle (2007). *Oracle Sensor Edge Server*. URL: https://www.oracle.com/technetwork/jp/content/index-090872-ja.html.

Park, C., W. Ryu, and B. Hong (2009). "RFID Middleware Evaluation Toolkit Based on a Virtual Reader Emulator". In: *Proceedings of the 1st International Conference on Emerging Databases*, pp. 1–4.

Paul, Reid (2007). *High cost slows RFID implementation*. URL: https://drugtopics.modernmedicine.com/drug-topics/news/clinical/pharmacy/high-cost-slows-rfid-implementation.

Power Pallet Recycling Center (2016). *Introducing Smart Pallets*. URL: https://powerpalletinc.com/introducing-smart-pallets/.

Quake Global, Inc. (2017). *EasyTAP™ | Quake Global*. URL: https://www.quakeglobal.com/products/easytap.

Ricoh Company Ltd. (2015). *RECO-Bridge IDR-1 V2 | RICOH*. URL: https://industry.ricoh.com/rfid/recobridge/.

Roberti, Mark (2015). *What Could Slow RFID Adoption?* URL: https://www.rfidjournal.com/articles/view?13693.

– (2017). *Is Retail Approaching the Tipping Point for RFID?* URL: https://www.iotjournal.com/articles/view?16669.

Ruiz-Sánchez, Miguel Á, Ernst W. Biersack, and Walid Dabbous (2001). "Survey and taxonomy of IP address lookup algorithms". In: *IEEE Network* 15.2, pp. 8–23. ISSN: 08908044. DOI: 10.1109/65.912716.

Saint-Andre, P. and J. Klensin (Apr. 2017). *Uniform Resource Names (URNs)*. RFC 8141. RFC Editor.

SAP AG. (2007). *SAP Auto-ID Infrastructure (SAP AII)*. URL: https://help.sap.com/saphelp_autoid2007/helpdata/en/99/6dfa4136696924e10000000a1550b0/frameset.htm?.

Schmidt, Loic et al. (2011). "DHT-based distributed ALE engine in RFID middleware". In: *2011 IEEE International Conference on RFID-Technologies and Applications, RFID-TA 2011*, pp. 319–326. ISBN: 9781457700279. DOI: 10.1109/RFID-TA.2011.6068656.

Seven & i Holdings Co., Ltd. (2015). *Annual Report 2015*. URL: https://www.7andi.com/en/ir/library/ar/2015.html.

Shannon, Claude E (1948). "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.July 1928, pp. 379–423. ISSN: 07246811. DOI: 10.1145/584091.584093. arXiv: 9411012 [chao-dyn]. URL: https://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf.

Sklower, Keith (1991). "A Tree-Based Packet Routing Table for Berkeley Unix". In: *USENIX Winter Conference*.

TAKAYA CORPORATION (2018). *RFID Products | TAKAYA*. URL: http://www.takaya.co.jp/product/rfid/.

The Ministry of Economy, Trade and Industry (METI) (2017). *Declaration of Plan to Introduce 100 Billion Electronic Tags for Products in Convenience Stores Formulated*. URL: https://www.meti.go.jp/english/press/2017/0418_003.html.

TOSHIBA TEC CORPORATION (2017). *Introducing RFID Reading Self-Checkout in GU of Fast Retailing Group (Japanese)*. URL: https://www.toshibatec.co.jp/release/20170420_01.html.

– (2018). *TOSHIBA TEC Products*. URL: http://www.toshibatec.com/products_overseas/.

Transcends, LLC. (2016). *RIFIDI Edge Server | Transcends*. URL: https://transcends.co/rifidi-edge-server/.

Tyco Retail Solutions (2016). *Macy's Leverages the Power of RFID to Fuel Successful Omni-Channel Fulfillment Strategy*. URL: https://www.tycoretailsolutions.com/Pages/ArticleDetail.aspx?ItemId=697.

Tyco Sensormatic (2018). *RFID Antennas & Readers | Tyco Sensormatic*. URL: https://www.sensormatic.com/products/inventory-intelligence/rfid-antennas-readers/.

W3C (June 2007). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Tech. rep. URL: https://www.w3.org/TR/wsdl20.

– (Apr. 2012). *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. Tech. rep. URL: https://www.w3.org/TR/xmlschema11-1/.

World Customs Organization (June 2004). *WCO UNIQUE CONSIGNMENT REFERENCE (UCR)*. Tech. rep.

Yilmaz, P.a. et al. (2000). "A trie-based algorithm for IP lookup problem". In: *Globecom '00 - IEEE. Global Telecommunications Conference. Conference Record (Cat. No.00CH37137)* 1, pp. 593–598. DOI: 10.1109/GLOCOM.2000.892085. URL: https://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=892085.

ZIH Corp (2018). *FX7500 Fixed RFID Reader | Zebra*. URL: https://www.zebra.com/us/en/products/rfid/rfid-readers/fx7500.html.