Data & Knowledge Engineering 70 (2011) 1070–1087

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

An approximate duplicate elimination in RFID data streams

Chun-Hee Lee [a], Chin-Wan Chung [b,*]

[a] Data Analytics Group, SAIT, Samsung Electronics, Yongin, 446–712, Republic of Korea
[b] Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, 305–701, Republic of Korea

ARTICLE INFO

Article history:
Received 6 March 2010
Received in revised form 16 July 2011
Accepted 18 July 2011
Available online 31 July 2011

Keywords:
Duplicate elimination
RFID
Bloom filter
Real-time DBs
Smart cards

ABSTRACT

The RFID technology has been applied to a wide range of areas since it does not require contact in detecting RFID tags. However, due to the multiple readings in many cases in detecting an RFID tag and the deployment of multiple readers, RFID data contains many duplicates. Since RFID data is generated in a streaming fashion, it is difficult to remove duplicates in one pass with limited memory. We propose one pass approximate methods based on Bloom Filters using a small amount of memory. We first devise Time Bloom Filters as a simple extension to Bloom Filters. We then propose Time Interval Bloom Filters to reduce errors. Time Interval Bloom Filters need more space than Time Bloom Filters. We propose a method to reduce space for Time Interval Bloom Filters. Since Time Bloom Filters and Time Interval Bloom Filters are based on Bloom Filters, they do not produce false negative errors. Experimental results show that our approaches can effectively remove duplicates in RFID data streams in one pass with a small amount of memory.

1. Introduction

Recently, due to the advancement of information technology, various kinds of data such as XML, RDF, and RFID data have been generated [18,9,5,8]. Especially, a large amount of RFID data has been generated in many environments since the RFID technology does not require contact in detecting RFID tags and therefore has been used in many areas such as business, military, and medical applications. The RFID adoption in Walmart is a typical RFID example in the business area.

However, the advantage of the RFID technology causes a new problem. Since an RFID tag is detected without contact, if an RFID tag is within a proper range from an RFID reader, the RFID tag will be detected whether we want to or not. Therefore, if RFID tags stay or move slowly in the detection region, much unnecessary data (i.e., duplicate RFID data) will be generated. On the other hand, if RFID tags move fast or many RFID tags move simultaneously in the detection region, one RFID reader may not be able to detect all of them. To prevent missing readings of RFID tags in such cases, several RFID readers are generally deployed in order to monitor a single location [1,2]. When several RFID readers detect one RFID tag at the same time, duplicate data is generated. Therefore, we cannot avoid the generation of duplicate RFID data in RFID applications.

An intelligent RFID reader with the processing capability can eliminate duplicate RFID data generated by the RFID reader. However, duplicate RFID data generated from multiple readers can not be removed with only the self-contained processing capability of RFID readers. Therefore, we need a technique to eliminate duplicate RFID data in the server (RFID middleware) that collects RFID data from various RFID readers.

Consider the example in Fig. 1. There are two RFID readers and two tags pass through the detection region. In this situation, Reader1 and Reader2 detect tags with the identifier ID1 and ID2. Each reader generates the detection information such as <tag ID, location of the reader, time>. Reader1 detects the tag with ID1 and generates RFID data <ID1, Loc1, 1>, <ID1, Loc1, 2>, <ID1, Loc1, 4>. However, the RFID data is duplicate data except <ID1, Loc1, 1>. Also, Reader1 generates RFID data <ID2, Loc1, 3> <ID2, Loc1, 5> for the tag with ID2 and <ID2, Loc1, 5> is duplicate data. In the same way, Reader2 generates RFID data and has duplicates as shown

* Corresponding author. Tel.: +82 42 350 3537; fax: +82 42 350 7737.
E-mail addresses: chunhee1.lee@samsung.com (C.-H. Lee), chungcw@kaist.edu (C.-W. Chung).

0169-023X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2011.07.007

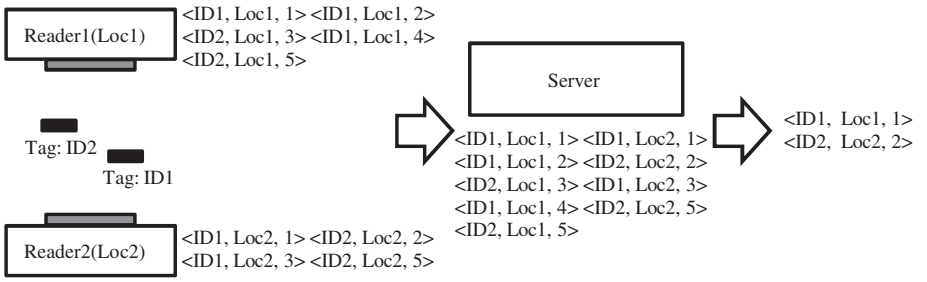


Fig. 1. An example for duplicate RFID data elimination.

in the lower part of Fig. 1. This is because a reader detects a tag continuously within the detection region. RFID data generated in each reader is sent to the server. Then, there are more duplicates in the server since two readers may detect the same tag. For example, <ID1, Loc1, 1> in Reader1 and <ID1, Loc2, 1> in Reader2 are duplicate. Therefore, the server removes duplicate RFID data as preprocessing and the result for preprocessing is <ID1, Loc1, 1> <ID2, Loc2, 2>. This problem seems simple for a small amount of RFID data.

However, the volume of RFID data is generally very big. Also, if a tag is attached to each item, the amount of data generated in a large retailer will exceed terabytes in a day [13]. And, RFID data is produced in a stream. It means that a duplicate elimination method should process data instantly with limited memory. It is difficult to design an exact duplicate elimination method. As an alternative, we can use approximate duplicate elimination techniques for RFID applications that do not require exact answers.

In such an application, we can consider a real-time analysis application for the movement of customers in a large department store. Each store in the department store has RFID readers. Each customer has a unique RFID tag. The manager wants the real-time analysis of the movement of customers such as the number of customers in each store and the store which has the maximum number of customers. For such an analysis, the central server in the department store should remove duplicates. In this environment, so much RFID data comes into the server simultaneously and there are duplicates. Especially, when a customer stays at the same location for a long time, it generates a large number of unnecessary duplicate data. In order to eliminate duplicates exactly, we need to keep all RFID data including duplicates in memory during a long period. Therefore, it is difficult to eliminate duplicates exactly in real-time using a small amount of memory relative to the amount of RFID data. In this application, the manager does not feel uncomfortable even if statistics with allowable errors are provided. Therefore, we propose approximate RFID duplicate elimination techniques in one pass with the limited memory.

Bloom Filters have been widely used as a very compact data structure with an allowable error. To manage RFID data streams with a small amount of memory, we devise Bloom Filter based approaches. However, since Bloom Filters are targeted for static data, we should adapt Bloom Filters to RFID data stream environments. We thus propose Time Bloom Filters as a straightforward adaptation of Bloom Filters. Since Time Bloom Filters are based on Bloom Filters, they do not generate false negatives which are duplicate data contained in the result after filtering. However, they may generate false positives that are non-duplicate data which are not contained in the result after filtering. We provide the false positive probability for Time Bloom Filters. Also, to reduce false positives for Time Bloom Filters, we devise Time Interval Bloom Filters using the concept of the interval.

Our contributions are as follows:

- *The Effective Duplicate Elimination Methods.* In data stream environments, it is not easy to design one pass duplicate elimination algorithm with limited memory. To design such an algorithm, we adapt Bloom Filters to RFID data stream environments. We propose effective approximate duplicate elimination methods, Time Bloom Filters and Time Interval Bloom Filters. They can eliminate duplicates in one pass with a small amount of memory.
- *Space Optimization for Time Interval Bloom Filters.* While the Time Bloom Filter stores one time field, the Time Interval Bloom Filter stores two time fields as a time interval. Therefore, the Time Interval Bloom Filter needs more space than the Time Bloom Filter. We devise a method to reduce space for the Time Interval Bloom Filter.
- *The Formulation of a Duplicate Elimination Problem.* Though many papers mention the duplicate elimination problem in RFID data, they do not formulate it rigorously. In this paper, we formulate the duplicate elimination problem in RFID data streams formally.
- *Parameter Setting.* In Time Bloom Filters and Time Interval Bloom Filters, the number of hash functions k affects errors. We propose a formula to find k and validate it using an experimental evaluation.

1.1. Organization

The rest of this paper is organized as follows. In Section 2, we present related work. In Section 3, we explain Bloom Filters as a preliminary, and in Section 4, we formalize the duplicate elimination problem in RFID data streams. We describe Time Bloom Filters and Time Interval Bloom Filters in Sections 5 and 6, respectively. In Section 7, we discuss parameter setting in Time Bloom Filters and Time Interval Bloom Filters. We measure the effectiveness of Time Bloom Filters and Time Interval Bloom Filters experimentally in Section 8. Finally, we conclude our work in Section 9.

2. Related work

As RFID tags have been fabricated at a very low cost, an enormous volume of RFID tags will be used in various places (e.g., retail stores, hospitals) in the near future. Therefore, a vast amount of RFID data will be generated. Chawathe et al.[8] discuss challenges for managing such an enormous volume of RFID data. Also, Bornhovd et al.[5] give an overview of the Auto-ID infrastructure (Device Layer/Device Operation Layer/Business Process Bridging Layer/Enterprise Application Layer) which integrates data from smart items (e.g., RFID, sensor) with existing business processes.

RFID-based item-tracking applications are typical RFID applications. To support item-tracking applications, some work has been done [14,13,3]. In [14], the bitmap datatype and its operations are introduced. By sharing the prefix of EPC,[1] a collection of EPCs is compactly represented in the bitmap datatype. However, [14] does not focus on query processing in an RFID-based item tracking. Gonzalez et al. [13] propose a new warehousing model, RFID-Cuboid, for RFID data compression and path-dependent aggregates. Also, to trace RFID tags efficiently, Ban et al. [3] propose the Time Parameterized Interval R-Tree. Cao et al.[6] present a distributed RFID stream processing system with the inference for tracking and monitoring. In addition, the works in [12,16,17] provide a tool to analyze a large amount of RFID data in the warehouse environment. Gonzalez et al. [12] propose FlowCube to analyze commodity flows at multiple levels. Lee and Chung [16,17] devise an effective path encoding scheme to answer path oriented queries.

As raw RFID data obtained from RFID readers misses readings and includes duplicate readings, there has been some work on RFID data cleansing [1,15,21]. Since RFID data from various sites is sent to the server, there exists heavy traffic in the network. In [1], to reduce network traffic, the edge processing is proposed. In the edge processing, data aggregation and smoothing, filtering, and grouping are performed as early as possible. The temporal smoothing filter that infers missing readings within a fixed time window is a general method to correct missing readings. However, it is difficult for users to decide the best time window size and the smoothing filter with a fixed time window size does not perform well due to the dynamics of RFID flows. Thus, to correct missing readings, Shawn et al.[15] use an adaptive window size according to the characteristics of RFID flows. In order to adapt the window size, they apply statistical sampling. Generally, data cleansing for RFID data streams is performed in a preprocessing step. However, Rao et al.[21] propose a deferred data cleansing method which is performed during query execution.

One of the most general and important cleaning of RFID data is to remove duplicate RFID data. There has been traditional research such as hash-based algorithms and sorting-based algorithms in duplicate elimination [11]. However, since those algorithms focus on duplicate elimination in stored data, they need to scan data many times. Thus, it is difficult to apply traditional duplicate elimination techniques to RFID data streams.

To eliminate duplicates for streaming data, approximate algorithms are proposed [20,10]. In [20], Metwally et al. use Bloom Filters for detecting duplicates in click streams. However, if click streams are generated continuously, all entries of the Bloom Filter will become 1 in the long run and then the Bloom Filter will be useless. In [10], Deng et al. solve the above problem using Stable Bloom Filters. In order to remove old data in a streaming environment, the Stable Bloom Filter sets cells corresponding to an input data to the maximum value and decreases values of randomly selected cells whenever data arrives. However, Stable Bloom Filters have false positive errors as well as false negative errors. In addition, the parameter setting in Stable Bloom Filters is complex and its solution is not always optimal. In [23], Wang et al. provide a method to remove duplicates over distributed data streams. The eager approach and the lazy approach are proposed to share Bloom Filters between the coordinator and the remote sites.

To eliminate duplicate RFID data, HashMerge is proposed in [2]. In HashMerge, a hash structure is used for an efficient duplicate elimination. However, it does not deal with the deletion operation which removes unnecessary data in the hash table. Also, the problem of RFID duplicate elimination is mentioned in [1,21,22] but they do not propose algorithms to eliminate duplicates. The authors in [19] use count Bloom Filters to remove duplicate RFID data. However, they consider duplicate elimination only at the reader level. Therefore, RFID readers should preprocess RFID data and send the preprocessed data to the server. In the case of RFID readers without the self-contained computing power, the approach cannot be applied. Also, the critical disadvantage of the approach is that it generates both false positives and false negatives.

While the above approaches focus on eliminating duplicate RFID data, [7] focuses on preventing the generation of duplicate RFID data. Duplicate RFID data generated in the same reader can be removed easily with the self-contained processing capability. However, duplicate RFID data generated in different readers cannot be removed with only the self-contained processing capability of the readers. To prevent the generation of duplicate RFID data in different readers, [7] deals with the redundant reader elimination. By removing the redundant readers which read the same tag simultaneously, [7] prevents the generation of duplicate RFID data. However, they assume writable tags on which some information can be written. Therefore, the method in [7] cannot be applied to the general cases.

3. Preliminary

Bloom Filters were originally proposed in [4]. The goal of Bloom Filters is to test whether any element is contained in a given set S using a small amount of memory. Although it is easy to test the membership of any element, if a set contains a large number of elements, we have to keep all elements of the set in memory and thus need a large amount of memory. Bloom Filters test the membership with allowable errors using a small amount of memory compared to the given set.

[1] Electronic Product Code(EPC) is a coding scheme of RFID tags to identify them uniquely.

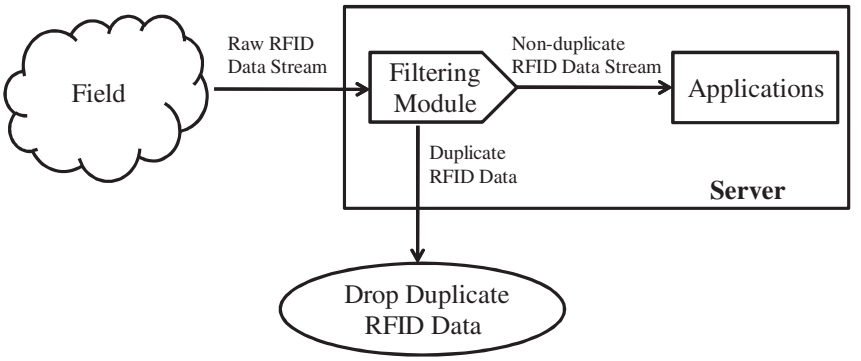


Fig. 2. The system architecture.

The Bloom Filter is based on a hash coding. It consists of an array of m bits and k hash functions $(h_1, h_2, \cdots, h_k)$. We assume that k hash functions address elements to range $\{0, \cdots, m-1\}$ with a uniform distribution and are independent from each other. Initially, all cells of the Bloom Filter are set to 0.

To represent a large number of elements in a set S using the Bloom Filter, the Bloom Filter sets cells, which correspond to $h_1(a)$, $h_2(a), \cdots, h_k(a)$, to 1 for each element a in S. Since one cell may be set to 1 by different elements, the Bloom Filter tests the membership of any element with errors.

To test whether an element a is contained in a set S using the Bloom Filter, we check k cells which correspond to $h_1(a), h_2(a), \cdots, h_k(a)$. If the value of at least one cell among k cells is 0, it is obvious that a is not in S. In this case, the Bloom Filter reports that a is not contained in S. If all values of k cells are 1, a is probably in S. The Bloom Filter reports that a is contained in S. In this case, there may exist false positive errors that report a is in S although it is not in S. However, in the Bloom Filter, there do not exist false negative errors that report that a is not in S although a is in S.

When the number of hash functions is k, the number of cells is m, and the number of elements in a set is n, the false positive probability f is $f = \left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k$ [4]. If m is large enough, $\left(1-\frac{1}{m}\right)^{kn} \approx e^{-kn/m}$. Thus, $f \approx (1-e^{-kn/m})^k$. When n and m are given, Bloom Filters can evaluate k to minimize f using derivatives. When $k = (ln2)\cdot\left(\frac{m}{n}\right)$, f is minimized. Therefore, Bloom Filters set k to $(ln2)\cdot\left(\frac{m}{n}\right)$ in order to minimize false positive errors.

4. Problem statement

The system architecture that we consider is depicted in Fig. 2. The RFID data stream generated in the field comes into the server. There are the Filtering Module and Applications in the server. Before the raw RFID data stream is moved to applications, it passes through the filtering module. With the filtering module, duplicate RFID data is removed from the raw RFID data stream. Applications receive non-duplicate RFID data in a stream. That is, in the filtering module, duplicate RFID data is immediately dropped from the raw RFID data stream.

RFID data is generated at many RFID readers continuously. RFID readers send RFID data to the server. Hence, the server receives a sequence of RFID data in a stream. RFID data consists of a tag identifier (i.e., EPC), the location of the RFID reader, and the detected time of the tag.

We define an RFID data stream formally as follows:

Definition 1. An RFID data stream S is a set $\{s_1, \cdots, s_n\}$,[2] where s_i consists of a triple $(TagID, Loc, Time)$ such that

- "TagID" is the electronic product code (EPC) of the tag and used for identifying the tag uniquely.
- "Loc" is the location of the reader which detects the tag.
- "Time" is the time of detecting the tag.

In an RFID data stream, RFID data x is considered as a duplicate if there exists $y(\neq x)$ in the RFID data stream such that $x.TagID = y.TagID$, and $x.Time - y.Time \leq \tau$, where τ is an application-specific positive value [2,21,22]. From the definition of the duplicate, we can find a non-duplicate RFID data stream by eliminating duplicates.

However, in a case that RFID data with the same $TagID$ is generated repeatedly at intervals that are less than or equal to τ, finding a non-duplicate RFID data stream is confusing. For example, consider an RFID data stream $S = \{s_1, s_2, s_3\}$, where $s_1 = (tag1, loc1, 5)$, $s_2 = (tag1, loc1, 10)$, and $s_3 = (tag1, loc1, 15)$ $(\tau = 8)$. From S, we know that s_2 is a duplicate because of s_1, and s_3 is a duplicate because of s_2. However, S arrives at the server in sequence. If s_2 arrives at the server, we can first remove s_2 from $\{s_1, s_2\}$. In that case, if s_3 arrives at the server, s_3 is not a duplicate since there does not exist s_2.

[2] Strictly speaking, an RFID data stream S is a sequence $<s_1, \cdots, s_n>$ but, in this section, we consider an RFID data stream as a set because elements in an RFID data stream contain the time information and a sequence is generated in time order.

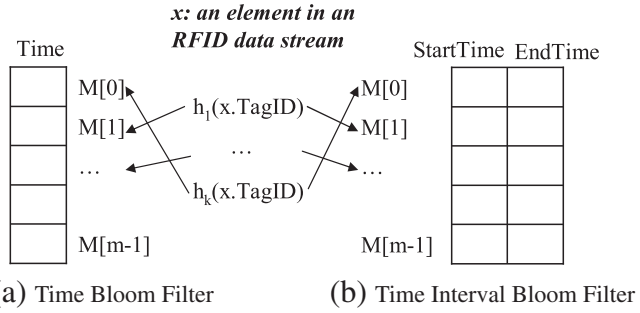


Fig. 3. The structures of the Time Bloom Filter and the Time Interval Bloom Filter.

Depending on the property of applications, a non-duplicate data stream for S can be defined differently. In this paper, a non-duplicate RFID data stream for S is considered as $\{s_1\}$ instead of $\{s_1, s_3\}$ since data with the same *TagID* that is generated repeatedly at intervals that are less than or equal to τ is generally useless.

We formulate the duplicate RFID data elimination problem formally with some definitions.

Definition 2. For RFID data x and y, x and y are connected in S if there exists $z \in S$ such that $x.TagID = y.TagID$, $z.TagID = x.TagID$, $|x.Time - z.Time| \leq \tau$ and $|z.Time - y.Time| \leq \tau$. x and y are connected in S also if x and z are connected in S and z and y are connected in S.

'Connected' is defined in order to remove duplicate RFID data of the same *TagID* that is generated repeatedly at intervals that are less than or equal to τ. If an RFID data stream contains two elements which have the same *TagID* and *Time*, one is an obvious duplicate. In this case, we consider the latter element in a sequence order as a duplicate. We focus on the elements with different *Time* to identify a duplicate. Using Definition 2, we define the duplicate formally in Definition 3.

Definition 3. RFID data x is a duplicate in S if there exists any $y(\neq x) \in S$ such that ($x.TagID = y.TagID$ and $x.Time - y.Time \leq \tau$) or ($x$ and y are connected in S and $y.Time \leq x.Time$).

The non-duplicate RFID data stream for an RFID data stream S is an RFID data stream after removing all duplicate RFID data in S. We can define the non-duplicate RFID data stream as the duplicate-free maximal set. The duplicate-free set and the duplicate-free maximal set are defined as follows:

Definition 4. A set $S'(\subset S)$ is called a duplicate-free set in S if there does not exists $x \in S'$ such that x is a duplicate in S.

Definition 5. The set $S'(\subset S)$ is a duplicate-free maximal set in S if S' is a duplicate-free set in S and for all $x \in S - S'$, x is a duplicate in S.

We want to find the duplicate-free maximal set S' (i.e., a non-duplicate RFID data stream) for an RFID data stream S. However, it is difficult to find the duplicate-free maximal set in data streams with a small amount of space. Since some applications allow errors, we consider a method to find an approximation of the duplicate-free maximal set. Especially, we limit allowable errors as false positive errors in this paper. Therefore, our problem is defined as follows: Given space m and a massive RFID data stream S, find a duplicate-free set $\hat{S}$ using space m such that $\left|\hat{S} - S'\right| / |S'|$ is minimized, where S′ is the duplicate-free maximal set in S.

5. Time Bloom Filters

To eliminate duplicates in RFID data streams, we first devise a simple approach based on Bloom Filters, called the Time Bloom Filter. From Definition 3, we know that RFID data x may not be a duplicate according to $x.Time$ even if there exists RFID data with the same *TagID* in an RFID data stream. Therefore, we can use the time information in detecting RFID duplicates. While each entry in the Bloom Filter is set to zero or one, each entry in the Time Bloom Filter is set to the detected time of an RFID tag. That is, the Time Bloom Filter uses an integer array instead of a bit array.

The Time Bloom Filter is depicted in Fig. 3(a). The Time Bloom Filter uses k independent hash functions $(h_1, h_2, \cdots, h_k)$ with range $\{0, \cdots, m-1\}$ like the Bloom Filter. The value of the i-th cell in the Time Bloom Filter is denoted by $M[i]$. In order to store RFID data x in the Time Bloom Filter, we find k cells that correspond to $h_1(x.TagID), \cdots, h_k(x.TagID)$. The Time Bloom Filter then sets k cells to the detected time of RFID data x (i.e., $x.Time$). If the cells are already set to the detected time of the previous RFID data, the Time Bloom Filter overwrites it with the detected time of the current RFID data.[3]

To know whether some RFID data x is a duplicate, we check k cells corresponding to $h_1(x.TagID), \cdots, h_k(x.TagID)$. If there exists at least one cell such that $x.Time - M[h_i(x.TagID)] > \tau$, we are sure that RFID data x is not a duplicate since duplicate data exists within τ time. In the

[3] We assume RFID data streams arrive at the server in time order for convenience.

Algorithm TBF(RFID_Data x)
begin
1. if $M[h_i(x.TagID)] \neq 0$ and
   $x.Time - M[h_i(x.TagID)] \leq \tau$ for all $i \in \{1, 2, \cdots, k\}$
2. drop x // x is a duplicate
3. else
4. send x to the application // x is not a duplicate
5. update $M[h_i(x.TagID)]$ to $x.Time$ for all $i \in \{1, 2, \cdots, k\}$

end

Fig. 4. Duplicate elimination algorithm of the Time Bloom Filter.

Time Bloom Filter, we can regard the condition $x.Time - M[h_i(x.TagID)] \leq \tau$ as 1 in the Bloom Filter and the condition $x.Time - M[h_i(x.TagID)] > \tau$ as 0 in the Bloom Filter. Note that the evaluation of the condition of the Time Bloom Filter changes as time goes by.

Fig. 4 shows the algorithm to eliminate duplicates in RFID data streams using the Time Bloom Filter. The algorithm can find the duplicate-free set with a small amount of memory. All cells in the Time Bloom Filter are initially set to 0. When RFID data x arrives at the server, x passes through the Time Bloom Filter. By the Time Bloom Filter, x will be filtered. In other words, x will be dropped or sent to the application. First, when x passes through the Time Bloom Filter, we check the condition $x.Time - M[h_i(x.TagID)] \leq \tau$ for all $i \in \{1,2,\cdots,k\}$ to know whether x is a duplicate or not (Line 1). Since all cells are initially set to 0, if the value of at least one cell is 0, x is not a duplicate regardless of the condition. If the condition $x.Time - M[h_i(x.TagID)] \leq \tau$ is satisfied and the value of $M[h_i(x.TagID)]$ is not 0 for all $i \in \{1,2,\cdots,k\}$, x will be probably a duplicate. Then, we drop it (Line 2). Otherwise, we send the non-duplicate data to the application (Lines 3–4). Finally, we update cells to $x.Time$ whenever x is a duplicate or not (Line 5). If we update cells only when that data is not a duplicate, we cannot remove duplicate data of the same *TagID* that is generated repeatedly at intervals that are less than or equal to τ.

Example 1. Fig. 5 shows the state of the Time Bloom Filter after an RFID data stream $S = \{s_1, s_2, s_3\}$ passes through the Time Bloom Filter, where $s_1 = (\text{ID1}, \text{Loc1}, 10)$, $s_2 = (\text{ID2}, \text{Loc2}, 120)$, and $s_3 = (\text{ID1}, \text{Loc1}, 130)$. Suppose hash functions are as shown in Fig. 5, the size of an array in the Time Bloom Filter is 8, k is 3, and τ is 100. To explain the Time Bloom Filter, we assume $h_2(ID1) = h_1(ID2)$.

Consider how the Time Bloom Filter operates for an RFID data stream $S = \{s_1, s_2, s_3\}$. When s_1 arrives at the Time Bloom Filter, it checks whether s_1 is a duplicate or not. Since all values in $M[0], M[5], and\ M[2]$ are initially 0, s_1 is not a duplicate. Therefore, s_1 is sent to the application. The Time Bloom Filter then sets $M[0], M[5], and\ M[2]$ to 10 in Fig. 5(a). When s_2 passes through the Time Bloom Filter, it is sent to the application since it is not a duplicate. The Time Bloom Filter sets $M[5], M[7], and\ M[3]$ to 120 as shown in Fig. 5(b). In the case of $M[5]$, the Time Bloom Filter sets it to 120 although 10 was set previously. When s_3 passes through the Time Bloom Filter, it is not a duplicate since $130 - M[0] > \tau$ and $130 - M[2] > \tau$ (M[0], and M[2] are values in Fig. 5(b)). s_3 is sent to the application and the Time Bloom Filter sets $M[0], M[5], and\ M[2]$ to 130 in Fig. 5(c). In this example, since all elements are not duplicates, the application receives all data.

We can derive the false positive probability for the Time Bloom Filter in a similar way as the Bloom Filter [4]. We provide the false positive probability for the Time Bloom Filter in Theorem 1.

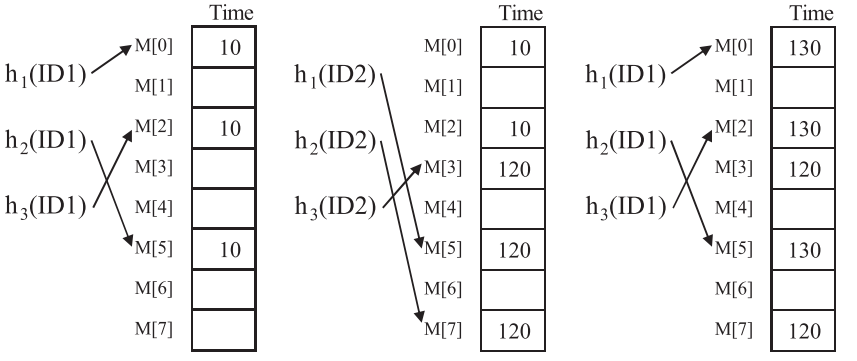


Fig. 5. The state of the Time Bloom Filter in Example 1.

Theorem 1. The false positive probability for the Time Bloom Filter is $\left(1-\left(1-\frac{1}{m}\right)^{kn'}\right)^k$, where k is the number of hash functions, m is the number of cells, and n' is the number of non-duplicate RFID data within τ time.

Proof. In order to derive the false positive probability, we assume that a non-duplicate RFID data stream comes to the Time Bloom Filter. In real applications, RFID data has many duplicates. However, duplicate RFID data sets the same cells in the Time Bloom Filter since they have the same TagID. Also, they set cells to a similar time since they are usually time clustered. Therefore, the state of the Time Bloom Filter after passing through the original RFID data is similar to that of the Time Bloom Filter after passing through non-duplicate RFID data removing duplicates from the original RFID data. Therefore, it is reasonable to assume that non-duplicate RFID data comes to the Time Bloom Filter.

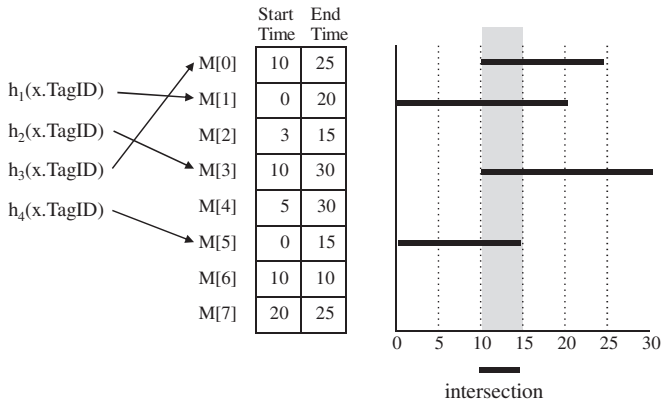
Consider, in the past, some RFID data passed through the Time Bloom Filter. Now, for any element x, we want to know the false positive probability. Since duplicates are defined within τ time, data that passed through the Time Bloom Filter before τ time is meaningless. Thus, we consider only data within τ time with respect to $x.Time$.

Let p_1 be the probability that $x.Time - M[h_i(x.TagID)] \le \tau$ for $1 \le i \le k$ and p_2 be the probability that x is a duplicate. Then,

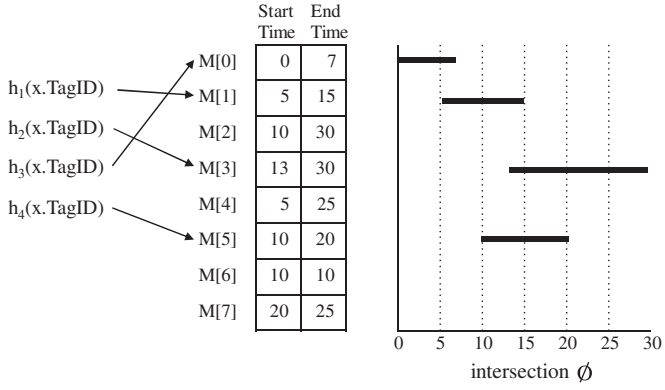
$$\text{The false positive probability for } x = p_1 - p_2$$

To derive p_1, we first derive the probability that $x.Time - M[u] \le \tau$ for some u. We assume that hash functions have unform distributions. For RFID data y within τ time, the probability that $h_i(y.TagID) \neq u$ for some i is $1-\frac{1}{m}$. The probability that $h_i(y.TagID) \neq u$ for all $i \in \{1,2,\cdots,k\}$ is $\left(1-\frac{1}{m}\right)^k$.

Since n' is the number of RFID data which comes to the Time Bloom Filter within τ time, the probability of $x.Time - M[u] > \tau$ is $\left(\left(1-\frac{1}{m}\right)^k\right)^{n'}$. The probability of $x.Time - M[u] \le \tau$ is $1-\left(1-\frac{1}{m}\right)^{kn'}$. Therefore, the probability p_1 that $x.Time - M[h_i(x.TagID)] \le \tau$ for all $i \in \{1,2,\cdots,k\}$ is $\left(1-\left(1-\frac{1}{m}\right)^{kn'}\right)^k$.



(a) The case where the intersection is not empty



(b) The case where the intersection is empty

Fig. 6. An example for the intersection of time intervals in the Time Interval Bloom Filter.

Since we assume that a non-duplicate RFID data stream comes to the Time Bloom Filter, p_2 is equals to 0. Therefore, the false positive probability for the Time Bloom Filter is $\left(1-\left(1-\frac{1}{m}\right)^{kn'}\right)^k$. □

6. Time interval bloom filters

In the previous section, we introduced Time Bloom Filters. However, since they are a simple extension of Bloom Filters, we devise Time Interval Bloom Filters to improve Time Bloom Filters. Since detection regions are distributed with proper ranges and RFID tags are detected only within or around detection regions, duplicate RFID data is location-clustered, and therefore time-clustered. That is, if some RFID tag is detected, RFID data with the same *TagID* will be detected many times during a period. Thus, we can represent the range of detected times of RFID data with the same *TagID* as a short time interval.

Fig. 3(b) shows the structure of the Time Interval Bloom Filter. The start time and the end time of the *i*-th cell are denoted by *M*[*i*].*StartTime* and *M*[*i*].*EndTime*, respectively. *StartTime* and *EndTime* are initially set to 0 and −1, respectively. Consider how the Time Interval Bloom Filter keeps StartTime and EndTime in a cell. For an easy explanation, we assume that the number of hash functions is 1 and one cell in the Time Interval Bloom Filter is set by only one TagID. When RFID data *x* with *TagID* = 1 arrives at the Time Interval Bloom Filter first, we set both *StartTime* and *EndTime* of the cell corresponding to *TagID* = 1 to *x.Time*. That is, the initial time interval is a time point. When another RFID data y with *TagID* = 1 arrives at the Time Interval Bloom Filter, we change only *EndTime* of the cell corresponding to *TagID* = 1 to *y.Time*. Thus, the Time Interval Bloom Filter keeps the time interval corresponding to *TagID* = 1 in the cell. For RFID data *x*, if *x.Time* − *EndTime* is more than τ, *x.Time* − *StartTime* is also more than τ. Such a time interval is useless. In this case, we set both StartTime and EndTime to *x.Time* again to initialize the time interval.

Since one cell may be set by various RFID data with different *TagID*, the time interval for each *TagID* may be mixed. However, the interval [*StartTime*,*EndTime*] in the cell includes detected times of all RFID data with *TagID* corresponding to the cell.

To know whether some RFID data *x* is a duplicate, we check whether the intersection of all time intervals corresponding to $h_1(x), h_2(x), \cdots, h_k(x)$ is empty. If some RFID data with *TagID* = *x.TagID* arrived at the Time Interval Bloom Filter, all time intervals corresponding to *x.TagID* should have included the detected time of that data and the intersection would have not been empty. Therefore, when the intersection is empty, we are sure that any RFID data with *TagID* = *x.TagID* did not arrive at the Time Interval Bloom Filter within τ time.

Fig. 6 illustrates how to determine whether some RFID data *x* is a duplicate. The coordinate on the right side of the Time Interval Bloom Filter of Fig. 6 represents the time intervals corresponding to RFID data *x*. In Fig. 6(a), the intersection for four time intervals is [10,15]. Therefore, it is not empty and *x* is considered as a duplicate. In Fig. 6(b), the intersection for four time intervals is empty. Therefore, the Time Interval Bloom Filter reports that *x* is not a duplicate.

Fig. 7 shows the algorithm to eliminate duplicates using the Time Interval Bloom Filter. If RFID data *x* arrives at the Time Interval Bloom Filter, we first check whether *x* is a duplicate or not. To do this, we check whether the intersection of intervals

```
Algorithm TIBF(RFID_Data x)
begin
1.  for(i = 1; i ≤ k; i++)
2.  {
3.      s_i = M[h_i(x.TagID)].StartTime
4.      e_i = M[h_i(x.TagID)].EndTime
5.  }
6.  intersectInterval = ⋂_{i=1}^{k} [s_i, e_i] // compute the intersection
7.  if intersectInterval ≠ φ and
        x.Time − intersectInterval.EndTime ≤ τ
8.      drop x // x is a duplicate
9.  else
10.     send x to the application // x is not a duplicate
11. for (i = 1; i ≤ k; i++) // update cells
12. {
13.     if M[h_i(x.TagID)].StartTime = 0 or
          x.Time − M[h_i(x.TagID)].EndTime > τ
14.     {
15.         M[h_i(x).TagID].StartTime = x.Time
16.         M[h_i(x).TagID].EndTime = x.Time
17.     }
18.     else
19.         M[h_i(x).TagID].EndTime = x.Time
20. }
end
```

Fig. 7. Duplicate elimination algorithm of the Time Interval Bloom Filter.

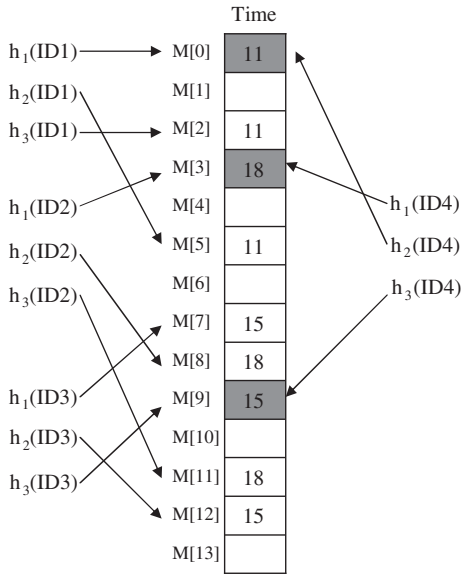


Fig. 8. The state of the Time Bloom Filter in Example 2.

corresponding to $h_1(x), h_2(x), \cdots, h_k(x)$ is not empty (Lines 1–7). If the intersection (intersectInterval in Line 6) is located in the time before $x.Time - \tau$ (i.e., $x.Time - intersectInterval.EndTime > \tau$), we regard it as empty since duplicates are defined within τ time. Therefore, when intersectInterval is not empty and $x.Time - intersectInterval.EndTime \leq \tau$, x is a duplicate and is dropped (Lines 7–8). Otherwise, x is sent to the application (Line 9–10).

We then update *StartTime* and *EndTime* (Lines 11–20). If $StartTime = 0$ (a case where data corresponding to the cell is first inserted) or $x.Time - EndTime > \tau$, we initialize both StartTime and *EndTime* to $x.Time$ (Lines 13–17). Otherwise, we update *EndTime* and extend time intervals (Lines 18–19).

Example 2. Consider an RFID data stream {(ID1, Loc1, 10), (ID1, Loc1, 11), (ID2, Loc2, 14), (ID3, Loc3, 15), (ID2, Loc4, 17), (ID2, Loc2, 18)}. Before explaining the operations of the Time Interval Bloom Filter, we explain those of the Time Bloom Filter. Fig. 8 shows the state of the Time Bloom Filter after the RFID data stream passes through the Time Bloom Filter. Suppose hash functions are as shown in Figs. 8 and 9, the size of an array in filters is 8, k is 3, and τ is 100. To explain the Time Interval Bloom Filter, we assume $h_1(ID1) = h_2(ID4)$, $h_1(ID2) = h_1(ID4)$, and $h_3(ID3) = h_3(ID4)$. If $(ID4, Loc4, 20)$ passes through the Time Bloom Filter in

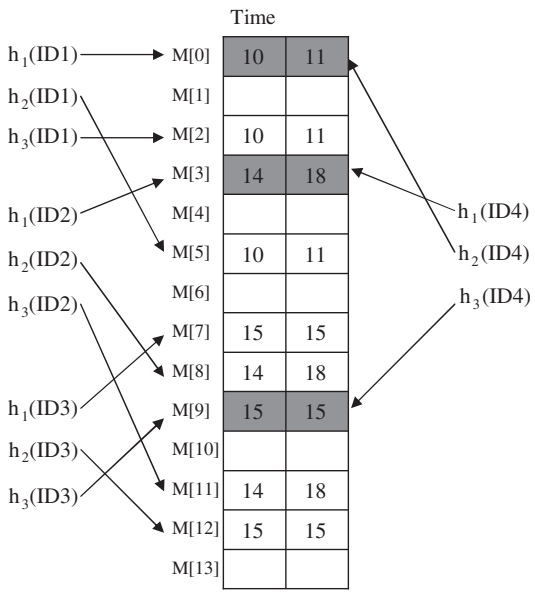


Fig. 9. The state of the Time Interval Bloom Filter in Example 2.

Fig. 8, the Time Bloom Filter reports that it is a duplicate since $20 - M[h_1(ID4)] \leq \tau$, $20 - M[h_2(ID4)] \leq \tau$ and $20 - M[h_3(ID4)] \leq \tau$. However, it is not a duplicate. That is, it is a false positive.

Meanwhile, the Time Interval Bloom Filter keeps the start time and the end time. Fig. 9 shows the state of the Time Interval Bloom Filter after the RFID data stream above passes through the Time Interval Bloom Filter. Consider the case where ($ID4$, $Loc4, 20$) passes through the Time Interval Bloom Filter in Fig. 9. The intersection of three intervals corresponding to *ID4* (i.e., [10,11], [14,18], [15]) is empty. Therefore, the Time Interval Bloom Filter reports that it is not a duplicate and it is really not a duplicate.

For the Time Interval Bloom Filter, it is difficult to derive the false positive probability since the false positive probability must be computed from the intersection of the time intervals. Instead of deriving the false positive probability for the Time Interval Bloom Filter, we provide Theorem 2. From Theorem 2, we know that the Time Interval Bloom Filter is better than the Time Bloom Filter in terms of the false positive probability.

Theorem 2. Consider the Time Bloom Filter and the Time Interval Bloom Filter which have the same number of cells and the same configuration for hash functions. Then, for any RFID data streams, the number of false positives of the Time Interval Bloom Filter is less than or equal to that of the Time Bloom Filter.

Proof. Assume that the same RFID data stream comes to the Time Bloom Filter and the Time Interval Bloom Filter. Since false positives are generated only when the Time Bloom Filter or the Time Interval Bloom Filter reports that a given data is a duplicate, we prove that for any element x, if the Time Interval Bloom Filter reports that x is a duplicate, then the Time Bloom Filter also reports that x is a duplicate.

Suppose that the Time Interval Bloom Filter reports that x is a duplicate. Then,

$$\begin{aligned} &intersectInterval \neq \phi \text{ and} \\ &x.Time - intersectInterval.EndTime \leq \tau. \end{aligned} \tag{1}$$

Since *intersectInterval* is not empty in the Time Interval Bloom Filter, all lines corresponding to x in the Time Interval Bloom Filter (i.e., $[s_i, e_i]$ for $1 \leq i \leq k$) include *intersectInterval*. Therefore,

$$\begin{aligned} &e_i \geq intersectInterval.EndTime \text{ for } 1 \leq i \leq k \\ &\quad (\text{also}, s_i \leq intersectInterval.StartTime) \end{aligned} \tag{2}$$

See how to update cells in the Time Bloom Filter and the Time Interval Bloom Filter (i.e., Line 5 in Fig. 4 and Lines 11–20 in Fig. 7). We can easily know that e_i in the Time Interval Bloom Filter is equal to m_i in the Time Bloom Filter, where e_i is defined in Line 4 of Fig. 7 and m_i is $M[h_i(x.TagID)]$ in the Time Bloom Filter.

Thus, by the Eq. (2)

$$m_i = e_i \geq intersectInterval.EndTime$$

By multiplying -1 in both sides

$$-m_i \leq -intersectInterval.EndTime$$

By adding $x.Time$ in both sides,

$$x.Time - m_i \leq x.Time - intersectInterval.EndTime$$

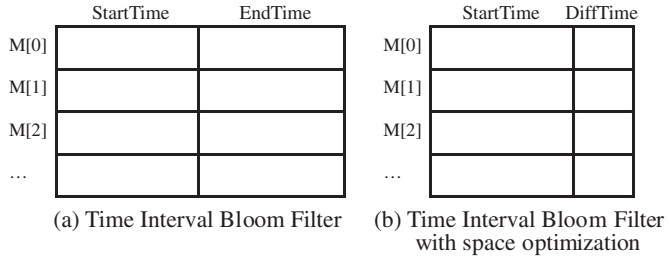
Since $x.Time - intersectInterval.EndTime \leq \tau$ by Eq. (1),

$$x.Time - m_i \leq \tau$$

That is, all conditions in Line 1 of Fig. 4 are satisfied. Therefore, the Time Bloom Filter reports that x is a duplicate. □

6.1. Space optimization

If the number of cells of the Time Bloom Filter is equal to that of the Time Interval Bloom Filter, the Time Interval Bloom Filter requires twice as much space as the Time Bloom Filter. However, since only the time interval within τ time is useful, we can optimize space for the Time Interval Bloom Filter. We keep StartTime and $EndTime - StartTime$ (*DiffTime*) instead of StartTime and EndTime as shown in Fig. 10. Since the space for DiffTime is less than that for EndTime, we can reduce the space. However, DiffTime may be more than τ while the algorithm to eliminate duplicates using the Time Interval Bloom Filter in Fig. 7 runs. Therefore, we revise the algorithm in order to keep DiffTime less than or equal to τ. Using the revised algorithm in Fig. 11, we can always keep DiffTime less than or equal to τ. We will explain it later. StartTime and EndTime require much space in reality compared with DiffTime, because they record in most cases month, date, hour, minute, and second. Since τ is a small value, we do not need much space to store DiffTime.



(a) Time Interval Bloom Filter (b) Time Interval Bloom Filter with space optimization

Fig. 10. Space optimization in the Time Interval Bloom Filter.

The duplicate elimination algorithm of the Time Interval Bloom Filter with space optimization is shown in Fig. 11. All StartTime and DiffTime are initially set to 0 and −1, respectively. Like the algorithm TIBF(), the algorithm TIBF_OPT() first checks whether *x* is a duplicate or not, then it drops x or sends x to the application (Line 1–10). In TIBF_OPT(), EndTime can be calculated easily from *StartTime* + *DiffTime* (Line 4).

To keep DiffTime less than or equal to τ, TIBF_OPT() update cells differently from TIBF() (Lines 11–28). When *StartTime* = 0 or *x.Time* − (*StartTime* + *DiffTime*) > τ, we set StartTime to *x.Time* and *DiffTime* to 0 in order to initialize time intervals in the Time Interval Bloom Filter (Lines 13–17). Otherwise, we change *StartTime* and *DiffTime*. In the case that *x.Time* − *StartTime* > τ, the part of the time interval is useless. To remove the useless part, we set StartTime to *x.Time* − τ and DiffTime to τ (Line 22–23). If *x.Time* − *StartTime* ≤ τ, we set only DiffTime to *x.Time* − *StartTime* (Lines 25–26).

Algorithm TIBF_OPT(RFID_Data x)
begin
1. for($i = 1; i \leq k; i++$)
2. {
3. $s_i = M[h_i(x.TagID)].StartTime$
4. $e_i = s_i + M[h_i(x.TagID)].DiffTime$
5. }
6. $intersectInterval = \bigcap_{i=1}^{k}[s_i,\ e_i]$ // compute the intersection
7. if $intersectInterval \neq \phi$ and
   $x.Time - intersectInterval.EndTime \leq \tau$
8. drop x // x is a duplicate
9. else
10. send x to the application // x is not a duplicate
11. for ($i = 1; i \leq k; i++$) // update cells
12. {
13. if $M[h_i(x.TagID)].StartTime = 0$ or
    $x.Time - (M[h_i(x.TagID)].StartTime +$
    $M[h_i(x.TagID)].DiffTime) > \tau$
14. {
15. $M[h_i(x.TagID)].StartTime = x.Time$
16. $M[h_i(x.TagID)].DiffTime = 0$
17. }
18. else
19. {
20. if $x.Time - M[h_i(x.TagID)].StartTime > \tau$
21. {
22. $M[h_i(x.TagID)].StartTime = x.Time - \tau$
23. $M[h_i(x.TagID)].DiffTime = \tau$
24. }
25. else
26. $M[h_i(x.TagID)].DiffTime =$
    $x.Time - M[h_i(x.TagID)].StartTime$
27. }
28. }
end

Fig. 11. Duplicate elimination algorithm of the Time Interval Bloom Filter with space optimization.

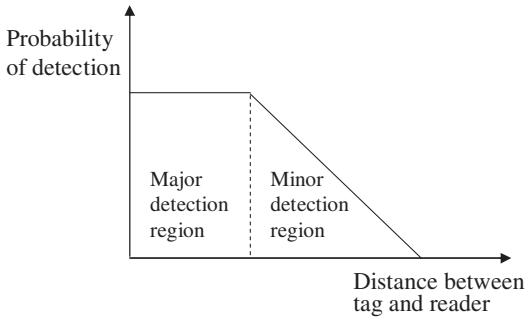


Fig. 12. The detection model.

7. Parameter setting

The false positive probability of Bloom Filters is $\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k}$ and k is set to $(ln2)\cdot\left(\frac{m}{n}\right)$ to minimize false positive errors, where m is the number of cells and n is the number of elements in a set. Similarly, we can find k for Time Bloom Filters to minimize false positive errors. The false positive probability of Time Bloom Filters is $\left(1-\left(1-\frac{1}{m}\right)^{kn'}\right)^{k}$, where n' is the number of non-duplicate RFID data within τ time. Therefore, when $k=(ln2)\left(\frac{m}{n'}\right)$, the false positive probability is minimized.

In this paper, we do not provide the false positive probability for Time Interval Bloom Filters. Instead, we provide Theorem 2. By Theorem 2, if the Time Bloom Filter and the Time Interval Bloom Filter have the same number of cells and the same configuration for hash functions, the Time Interval Bloom Filter is better than the Time Bloom Filter in terms of false positive errors. If we set k in the Time Interval Bloom Filter to $(ln2)\left(\frac{m}{n'}\right)$ like the Time Bloom Filter, the Time Bloom Filter and the Time Interval Bloom Filter have the same configuration for hash functions. Therefore, when we set k in the Time Interval Bloom Filter to $(ln2)\left(\frac{m}{n'}\right)$, we guarantee that the Time Interval Bloom Filter is better than the Time Bloom Filter if the Time Bloom Filter and the Time Interval Bloom Filter have the same number of cells.

Thus, for both Time Bloom Filters and Time Interval Bloom Filters, we set k to $(ln2)\left(\frac{m}{n'}\right)$, where n' is the number of non-duplicate RFID data within τ time.

8. Experiments

In order to measure the effectiveness of our approaches, we conducted an experimental evaluation. There are some works which deal with the duplicate RFID data elimination problem, however, to the best of our knowledge, our work is the first to systematically deal with an approximate duplicate elimination in RFID data streams. Therefore, we compared our two approaches, the Time Bloom Filter and the Time Interval Bloom Filter. Also, we included Bloom Filters to observe their behavior when applied to the RFID data stream. We experimented on a Pentium 3GHz PC with 1 GB main memory using Java.

8.1. Data sets

Since there does not exist a well-known RFID data set, we made a synthetic data generator similar to [15]. To simulate the detection of an RFID tag, we used the detection model in [15].

If the distance between a tag and an RFID reader is far enough, the tag is not detected. As the tag approaches the RFID reader, the tag is detected with the probability proportional to the distance. The region is called the minor detection region as shown in Fig. 12 [15]. When the tag and the RFID reader are in a close distance, the detection probability is constant. The region is called the major detection region.

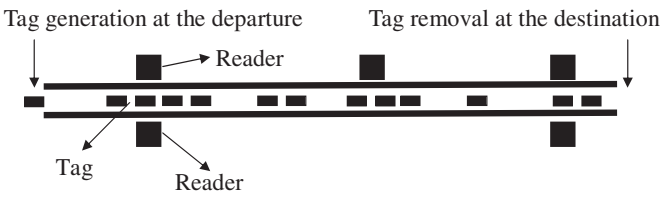


Fig. 13. The tag generation model.

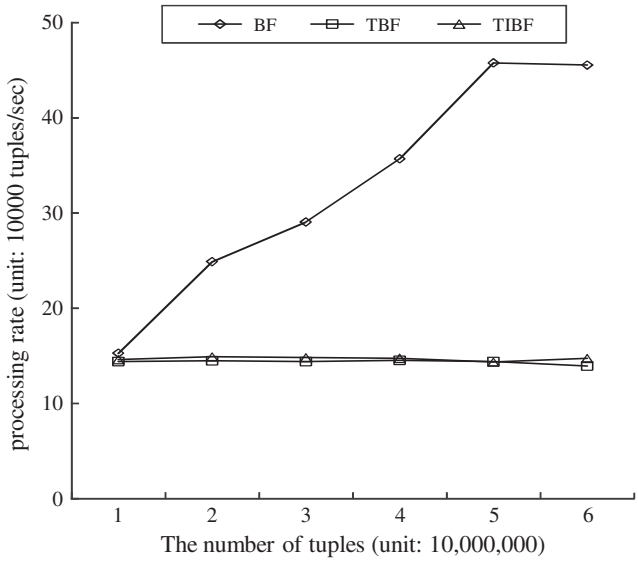


Fig. 14. The processing rate for SData according to the number of tuples.

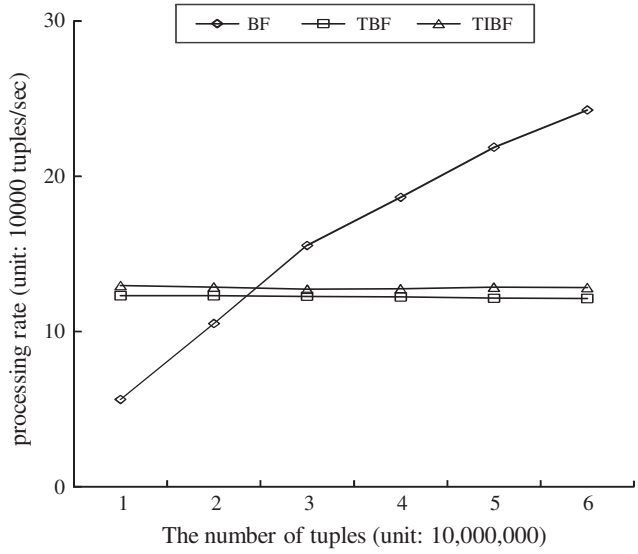


Fig. 15. The processing rate for MData according to the number of tuples.

Since our goal is to test whether duplicate data is eliminated effectively, we assume that tags move in a straight line as shown in Fig. 13. Tags are generated at only the departure and then move in a straight line with the velocity that is assigned randomly when tags are generated. We assign the same velocity for tags generated at the same time since tags generally move together in real applications. If a tag reaches the destination, the tag will disappear. Also, in a straight line, there are many detection locations and there may exist multiple readers at detection locations in order to improve the detection ratio. We generate 10^7, 2×10^7, 3×10^7, 4×10^7, 5×10^7, and 6×10^7 tuples both in the case where there exists a single RFID reader at a detection location and in the case where there exist three RFID readers at a detection location. We call the former data SData and the latter data MData. MData has more duplicate data than SData.

8.2. Experimental results

We evaluate the Time Bloom Filter and the Time Interval Bloom Filter[4] with respect to a processing rate and an error rate. We show experimental results according to the number of tuples in Section 8.2.1 and those according to the space size in Section 8.2.2.

[4] The Time Interval Bloom Filter in this section uses space optimization.

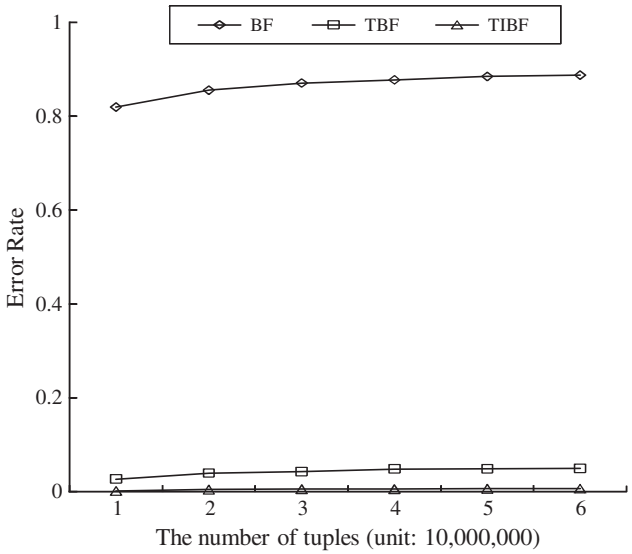


Fig. 16. The error rate for SData according to the number of tuples.

8.2.1. Experiments according to the number of tuples

Figs. 14 and 15 show the processing rate according to the number of tuples. A processing rate is the number of tuples that are processed in a unit time. We fix the amount of allowable memory to 4×10^7 bits. The Bloom Filter (BF) shows the best processing rate among all approaches for SData as shown in Fig. 14 since it has the simplest structure. In the Bloom Filter, the processing rate increases when the number of tuples increases. k in the Bloom Filter decreases when the number of tuples increases because k is set to $(ln2)\cdot\left(\frac{m}{n}\right)$. If k is small, we will set and scan a few cells in the Bloom Filter. Thus, in the Bloom Filter, the processing rate increases as the number of tuples increases. However, in the Time Bloom Filter (TBF) and the Time Interval Bloom Filter (TIBF), since k is not affected by the number of tuples, the processing rate is constant. In MData (Fig. 15), the processing rate shows a similar result as SData (Fig. 14). The processing rates in the Time Bloom Filter and the Time Interval Bloom Filter are constant and that in the Bloom Filter increases as the number of tuples increases. Also, in the case of the Bloom Filter, the processing rate for MData is less than that for SData because k for MData is larger than k for SData. Note that k is set to $(ln2)\cdot\left(\frac{m}{n}\right)$, where m is the number of cells and n is the number of distinct elements in a set.

To validate the effectiveness of the Time Bloom Filter and the Time Interval Bloom Filter, we evaluate the error rate according to the the number of tuples. The error rate of the Bloom Filter is much worse than those of other approaches for both SData (Fig. 16) and MData (Fig. 17) as we expected. The error rate of the Time Interval Bloom Filter is the best among all approaches and is less

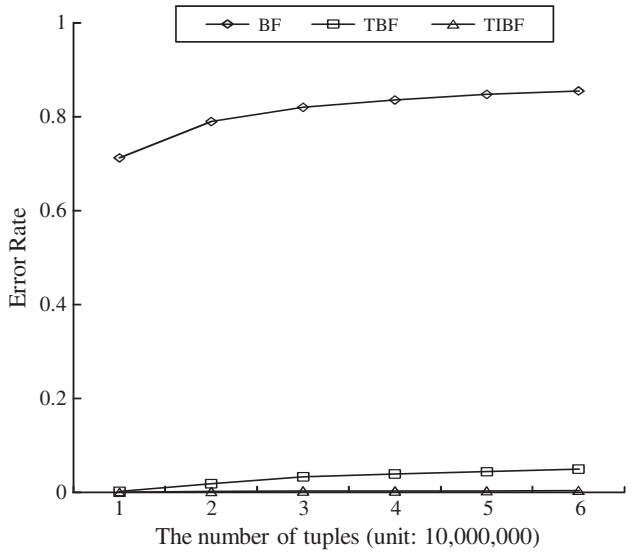


Fig. 17. The error rate for MData according to the number of tuples.

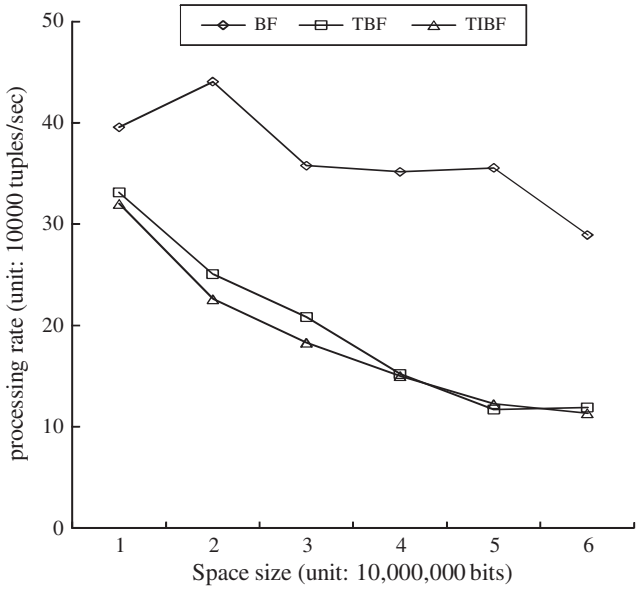


Fig. 18. The processing rate for SData according to the space size.

than 0.007% in all cases. The error rate of the Time Bloom Filter is a little worse than that of the Time Interval Bloom Filter. The error rates of all approaches increase as the number of tuples increases.

8.2.2. Experiments according to the space size

We evaluate the processing rate and the error rate according to space size given the number of tuples is 4×10^7. Figs. 18 and 19 show how the processing rate changes as the space size increases. In the Bloom Filter, the Time Bloom Filter, and the Time Interval Bloom Filter, since k is set proportional to the space size, the processing rate shows a tendency to decrease as the space size increases.

While the processing rate of the Bloom Filter for SData (Fig. 18) is much better than other approaches, the processing rate of the Bloom Filter for MData (Fig. 19) is a little better than the Time Bloom Filter and the Time Interval Bloom Filter. This is because k of the Bloom Filter for MData increases much more than that for SData compared to the Time Bloom Filter and the Time Interval Bloom Filter.

The error rate according to the space size is shown in Figs. 20 and 21. In the Bloom Filter, the error rate stays almost constant as the space increases since most of entries in the Bloom Filter are set to 1. However, the error rate of the Time Bloom Filter, the Time

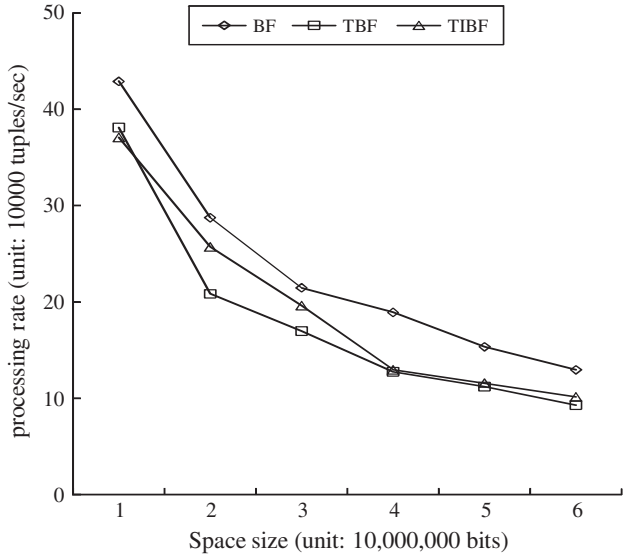


Fig. 19. The processing rate for MData according to the space size.

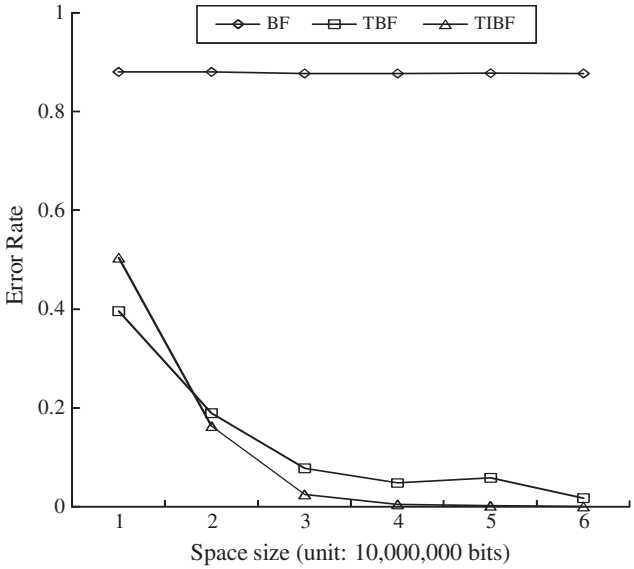


Fig. 20. The error rate for SData according to the space size.

Interval Bloom Filter decreases considerably as the space increases. As shown in Figs. 20 and 21, the error rate of the Time Interval Bloom Filter is better than other approaches in many cases. Since the Bloom Filter cannot change the cell dynamically, the error rate is high even for the case of the small number of tuples.

8.2.3. Optimal k

Like the Bloom Filter, k affects the error rate of the Time Bloom Filter and the Time Interval Bloom Filter. To investigate whether the setting of k proposed in Section 7 is valid, we evaluate the error rate as k changes from 1 to 10. Figs. 22 and 23 show the error rate according to k given that space size is 3×10^7 bits and the number of tuples is 3×10^7. In Figs. 22 and 23, the thick circle means the k value evaluated by the formula in Section 7 and the double circle means the k value evaluated by applying the formula of Bloom Filters.

In Figs. 22 and 23, all k values corresponding to the thick circles are considered good choices in terms of error rates. However, the error rates corresponding to the double circles are much worse than the optimal error rates. We evaluated the error rate according to k for various space sizes and data sizes. As a result, we conclude that the proposed k value in Section 7 is considered as a good choice in most cases although it is not always optimal.

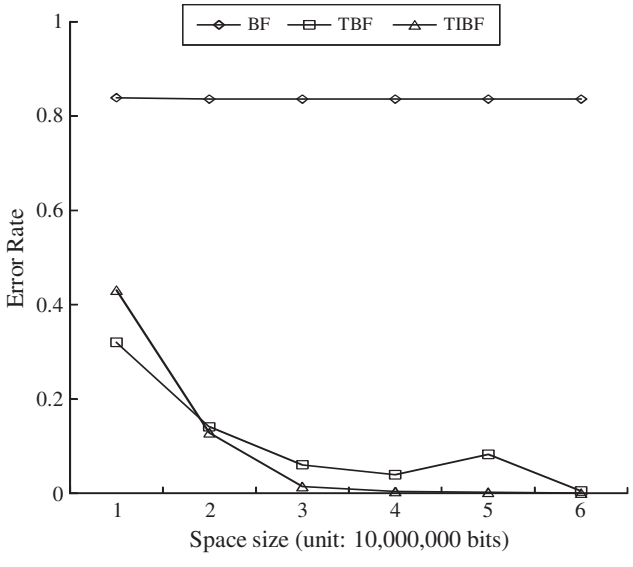


Fig. 21. The error rate for MData according to the space size.

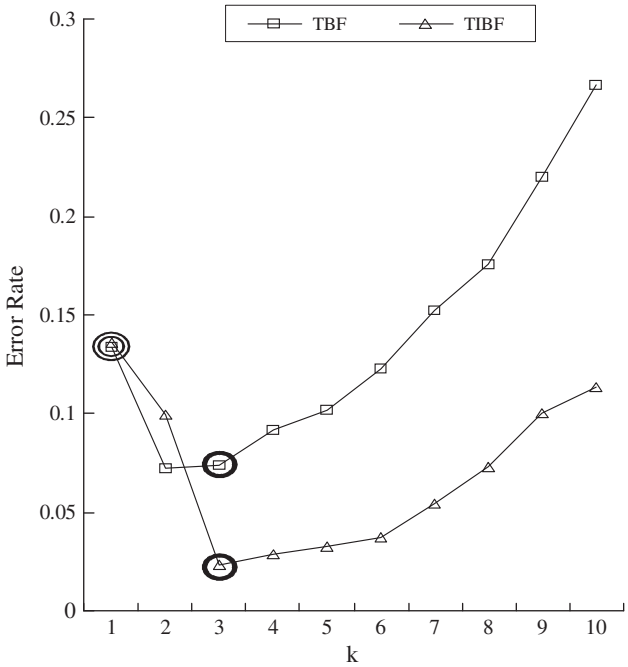


Fig. 22. The error rate for SData according to *k*.

9. Conclusion

Generally, RFID data streams include numerous duplicate data. Since RFID data is produced in a stream, it is difficult to eliminate duplicate RFID data in one pass with a small amount of memory. Thus, we propose approximate duplicate elimination methods, Time Bloom Filters and Time Interval Bloom Filters, based on Bloom Filters. In the Time Bloom Filter, we replace a bit array in the Bloom Filter with an array of time information since the definition of duplicates in RFID data is related to the detected

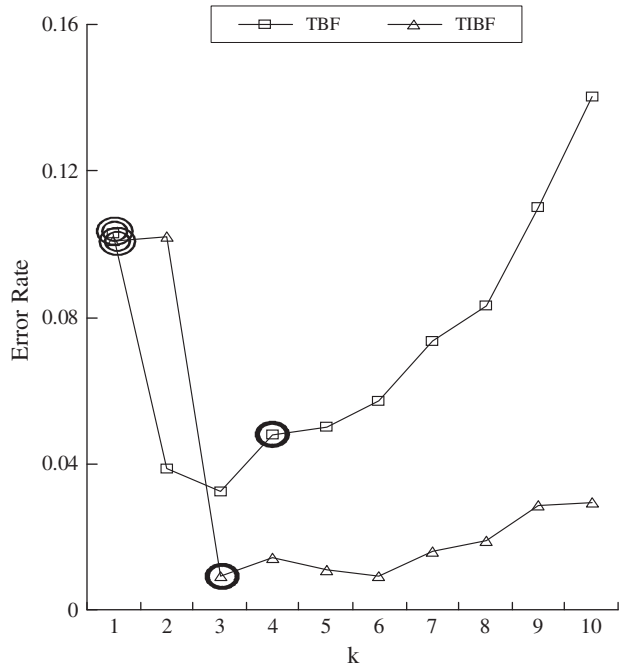


Fig. 23. The error rate for MData according to *k*.

time. Also, to reduce false positives, the Time Interval Bloom Filter uses a time interval. Finally, experimental results show that the proposed approaches can remove duplicate RFID data in one pass with a small error.

Acknowledgments

We would like to thank the editor and anonymous reviewers. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0000377).

References

[1] Manage data successfully with RFID anywhere edge processing, http://www.ianywhere.com/developer/rfidanywhere/rfidanywhereedgeprocessing.pdf.
[2] Y. Bai, F. Wang, P. Liu, Efficiently filtering RFID data streams, VLDB Workshop on Clean Databases, 2006.
[3] C. Ban, B. Hong, D. Kim, Time parameterized interval R-tree for tracing tags in RFID systems, DEXA, 2005, pp. 503–513.
[4] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communication of the ACM 13 (7) (1970) 422–426.
[5] C. Bornhövd, T. Lin, S. Haller, J. Schaper, Integrating automatic data acquisition with business processes — experiences with SAP's Auto-ID infrastructure, VLDB, 2004, pp. 1182–1188.
[6] Z. Cao, C. Suttony, Y. Diao, P. Shenoy, Distributed inference and query processing for RFID tracking and monitoring, PVLDB 4 (5) (2011) 326–337.
[7] B. Carbunar, M.K. Ramanathan, M. Koyutürk, C. Hoffmann, A. Grama, Redundant reader elimination in RFID systems, IEEE SECON, 2005, pp. 176–184.
[8] S.S. Chawathe, V. Krishnamurthy, S. Ramachandran, S. Sarma, Managing RFID data, VLDB, 2004, pp. 1189–1195.
[9] A. Chebotko, S. Lu, X. Fei, F. Fotouhi, RDFProv: a relational RDF store for querying and managing scientific workflow provenance, Data & Knowledge Engineering 69 (8) (2010) 836–865.
[10] F. Deng, D. Rafiei, Approximately detecting duplicates for streaming data using stable bloom filters, SIGMOD, 2006, pp. 25–36.
[11] H. Garcia-Molina, J.D. Ullman, J. Widom, Database System Implementation, Prentice Hall International, Inc., Upper Saddle River, New Jersey, 2000, p. 07458.
[12] H. Gonzalez, J. Han, X. Li, FlowCube: constructuing RFID FlowCubes for multi-dimensional analysis of commodity flows, VLDB, 2006, pp. 834–845.
[13] H. Gonzalez, J. Han, X. Li, D. Klabjan, Warehousing and Analyzing Massive RFID Data Sets, ICDE, 2006.
[14] Y. Hu, S. Sundara, T. Chorma, J. Srinivasan, Supporting RFID-based item tracking applications in Oracle DBMS using a bitmap datatype, VLDB, 2005, pp. 1140–1151.
[15] S.R. Jeffery, M.N. Garofalakis, M.J. Franklin, Adaptive cleaning for RFID data streams, VLDB, 2006, pp. 163–174.
[16] C.-H. Lee, C.-W. Chung, Efficient storage scheme and query processing for supply chain management using RFID, SIGMOD, 2008, pp. 291–302.
[17] C.-H. Lee, C.-W. Chung, RFID data processing in supply chain management using a path encoding scheme, IEEE Transactions on Knowledge and Data Engineering (TKDE) 23 (5) (2011) 742–758.
[18] J. Lu, X. Meng, T.W. Ling, Indexing and querying XML using extended Dewey labeling scheme, Data & Knowledge Engineering 70 (1) (2011) 35–59.
[19] H. Mahdin, J. Abawajy, An approach to filtering duplicate RFID data streams, Communications in Computer and Information Science 124 (2010) 125–133.
[20] A. Metwally, D. Agrawal, A.E. Abbadi, Duplicate detection in click streams, WWW, 2005, pp. 12–21.
[21] J. Rao, S. Doraiswamy, H. Thakkar, L.S. Colby, A deferred cleansing method for RFID data analytics, VLDB, 2006, pp. 175–186.
[22] F. Wang, P. Liu, Temporal management of RFID data, VLDB, 2005, pp. 1128–1139.
[23] X. Wang, Q. Zhang, Y. Jia, Efficiently filtering duplicates over distributed data streams, International Conference on Computer Science and Software Engineering (CSSE), 2008, pp. 631–634.

Chun-Hee Lee received a PhD degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Korea, in 2010. His research interests include sensor network, stream data management, and graph databases.

Chin-Wan Chung received a B.S. degree in electrical engineering from Seoul National University, Korea, in 1973, and a Ph.D. degree in computer engineering from the University of Michigan, Ann Arbor, USA, in 1983. From 1983 to 1993, he was a Senior Research Scientist and a Staff Research Scientist in the Computer Science Department at the General Motors Research Laboratories (GMR). Since 1993, he has been a professor in the Department of Computer Science at the Korea Advanced Institute of Science and Technology (KAIST), Korea. He was in the program committees of major database and Web conferences including ACM SIGMOD, VLDB, IEEE ICDE, and WWW. He was an associate editor of ACM TOIT, and is an associate editor of WWW Journal. He will be the general chair of WWW2014. His current research interests include the semantic Web, social networks, mobile Web, sensor networks and stream data management, and multimedia databases.