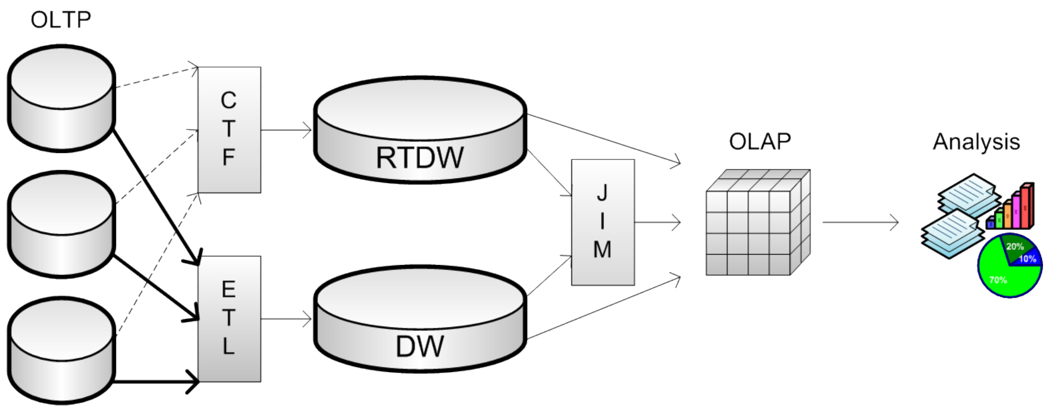
A Real-Time Data Warehouse Solution for Analysis on Indoor Tracking Data

Department of Computer Science @ Aalborg University



Group d621b
Jonas T. Hansen
Stig Jørgensen

Title:
A Real-Time Data Warehouse Solution for Analysis on Indoor Tracking Data

Project Period:
SW10: February 1st to July 31st, 2009

Project Group:
Computer Science, d621b

Members:
Jonas T. Hansen
Stig Jørgensen

Supervisor:
Hua Lu

Copies:
5

Numbered Pages:
57

Total Pages:
66

Appendices:
A-C

Abstract:

This project concerns the design and implementation of a real-time data warehouse for analysis on tracking data collected at Copenhagen Airports A/S. The project is done in cooperation with BLIP Systems A/S, who has a Bluetooth based tracking system installed in Copenhagen Airports A/S. First, we describe how BLIP Systems A/S stores the collected tracking data and propose a data warehouse design that enables answering some business intelligence related questions. As a part of the capture, transform, and flow, some data cleansing has been performed on the source data. Additionally, we introduce two case-specific algorithms, *ValidMoves* and *BounceDetection*, which normalizes the tracking data. By utilizing custom applications and Microsoft Analysis Server, we address some of the questions presented and provide answers through views and GUIS. We conclude on the project and suggests directions for further work. Last, the report contains a summary in Appendix C.

Preface

This master thesis extends and builds upon the work and experiences made in the pre-master thesis report – *A Data Warehouse Solution for Analysis on Indoor Tracking Data* [11]. The thesis is written by Jonas T. Hansen and Stig Jørgensen under supervision of Hua Lu at the Database and Programming Technologies research unit located at Aalborg University. The data basis for this thesis is provided by BLIP Systems A/S [22], which is a software developing house located in North Jutland. BLIP Systems A/S is a company that specializes in Bluetooth solutions for Bluetooth- marketing and tracking software systems. BLIP Systems A/S will henceforth be referenced to merely as BLIP.

Whereas the pre-master thesis focused upon building a traditional historical Data Warehouse in order to answer a series of Business Intelligence questions, this master thesis will focus on designing and implementing a Real-Time Data Warehouse. Emphasis will be put on the process of capturing, transforming, and feeding data to the real-time data warehouse. The solution is inspired by the pre-master thesis [11]. Section 3 provides the basics for implementing the RTDW, whereas Section 4 covers the actual implementation of the system, which improves the algorithms introduced in our pre-master thesis and adapts them to a real-time environment, and Section 5 presents the results we have obtained by testing the different parts of the system.

Lastly, the report includes a Summary in Appendix C which covers the report in greater detail than the abstract.

Contents

1 Introduction

Traditional historical data warehouses are typically loaded at fixed intervals, typically once every day, week, or month depending on the business needs of knowledge workers and the amount of data present in the operational systems. Data is usually loaded into the Data Warehouse (DW) at night or more specifically when the usage load is at a minimum to avoid consuming the much needed computational power of the Online Analytical Processing (OLAP) system. Loading the potentially enormous amount of data at off-hours allows the system to process the data so it is ready for analysis when needed.

Commonly, traditional data warehouses provide a means for Business Intelligence (BI) analysts, or knowledge workers, to capture the data from the past, analyze it in every detail, and use the knowledge gained to try to predict the future and plan accordingly - we define this process as strategic decision making [2]. What about knowledge of the present - could this knowledge not be equally, or even more, important? This is where the Real-Time Data Warehouses (RTDW) comes into play. RTDWs differ from traditional DWs in that they are continuously informed of any changes in the operational systems and therefor represent the environment in its current state. Being able to perform complex analysis of a domain in its current state can be useful in many ways, and means that immediate tactical actions can be taken when necessary - we define this as tactical decision making [2]. Needless to say, not all domains can benefit of RTDWs in the same degree as others might. An example of where RTDWs can be put to good use are:

Within the financial section, institutions lose millions of dollars each year due to credit card fraud. It is possible to detect fraud by analyzing credit card transactions for certain types of credit card use patterns. If these patterns can be identified, the bank can ban the credit card, preventing it from being exploited numerous times and thereby reduce losses dramatically. This analysis has to be done in real-time as stolen cards are typically abused several times within a short period of time or until the card has been banned. Waiting for historical data to be entered into the data warehouse before an attempt to detect fraud can be made, would give the criminals plenty of time to conduct their business before the fraud would even be discovered [13].

It is important to realize that real-time data should not replace, but complement, historical data. The real-time data can be used to predict and/or influence sales of a business in the near future or discover traffic jams in the present, which allows knowledge workers to react efficiently to changes or remedy unfavorable situations in the domain - e.g. an online shopping site can recommend certain products for a user, based either on the users historical purchases and ratings of products or on the products browsed in the current session. This illustrates the difference between strategic and tactical decision making.

We will use the BLIP case from the pre-master thesis, mentioned in the preface, and concentrate on building a real-time data warehouse that will enable us to answer questions which we were unable to answer with a traditional data warehouse. We will strive to emphasize the changes that comes with this shift in paradigms and primarily focus on the process of capturing, transforming, and feeding data in real-time to the RTDW.

1.1 Goals

This master thesis will deal with creating a case-specific real-time data warehouse using BLIPs data and business case. This leads to a number of goals we will strive to fulfill and BI questions to answer through the master thesis. When designing the RTDW, CTF, and user interfaces, the needs of these goals will have to be taken into consideration in order to improve the data quality, analysis responsiveness, and thereby the quality of the reports generated.

The goals are:

1. *Creating a fast, flexible, and durable real-time data warehouse.*
   A lot of strain is put on the OLAP system because data is continuously being fed to the real-time data warehouse, and query results are expected to be up-to-date at all times. This means that the RTDW must be designed in such a way that dimensions and facts are stored and aggregated at the right time and in the most optimal way to yield the best query response time possible. This provides the base for a solid real-time data warehouse and thereby effective and efficient analysis.

2. *CTF data throughput.*
   To ensure that the data throughput of the CTF is optimal, faulty data must be discarded as early as possible. This reduces the overall amount of data to be processed. The CTF should be able to process records fast enough to ensure that the data flow is *real-time*. Speed, however, should not be gained at the expense of data quality.

3. *CTF data output quality.*
   This is an issue we encountered during the pre-master thesis. Because of BLIPs data gathering method, many records can be classified as *noise* - what we call *bouncing records*, but due to the fact of the CTF being real-time, there is no way to pre-empt the records and clean them before they have been inserted into the RTDW. Therefor, to increase overall data quality, we want to allow the CTF to make changes to certain records in the warehouse. Changing records that have already been inserted into the RTDW does not compromise the business requirements.

The BI questions are:

1. *Are there any congested areas in the airport?*
   Knowing exactly which areas or zones that are congested, and how congested they are, can be used to deploy personnel and heighten security to try to remedy the situation in the affected areas.

2. *Are there any congestions forming?*
   Closely related to the previous BI question, this knowledge gives the airport a preemptive advantage in dealing with congestion.

3. *Are there any passengers that are in risk of not reaching their gate in time?*
   Passengers not shoving up at the gate in time is a common reason for flight delays. By knowing a passengers location, departure gate, and departure time, an application can be used to predict whether or not a passenger can reach his gate in time, giving the staff or a system the opportunity to take action accordingly.

To achieve the goals and answer the questions posed, we need to propose a set of algorithms and design a data warehouse that is based on an architecture that performs optimally within the scope of real-time. To do this, we decide to learn from other peoples mistakes and list the most common pitfalls of RTDW construction.

1.2 Pitfalls

Based on Stephen Brobsts *Ten Mistakes to Avoid When Constructing a Real-Time Data Warehouse* [2], we will list the most common mistakes made, when designing and implementing real-time data warehouses. A short description of each pitfall will be made. Throughout the report we will weigh our decisions up against the pitfalls listed here in order to present valid arguments for our choices.

1. **Focusing on real-time rather than ”right-time:”**
   The first common mistake is spending an enormous amount of resources on implementing real-time feeds when ”right-time” could suffice. The idea of right-time covers that data freshness service level agreements are driven by business needs rather than the real-time hype. A right-time implementation, conforming to the Service Level Agreements (SLAs) [24], can provide a more cost- and time-effective solution for the business.

2. **Confusion between bookkeeping, decision making, and action taking:**
   The inability to distinguish between the operational bookkeeping systems, the decision making systems and action taking systems. The operational systems are often transactional oriented with a high write to read ratio, whereas the decision environment has a high read to write ratio. Action taking is the cooperation between the operational systems and the decision making systems, where decisions are acted out by entering or changing data in the transactional systems.

3. **Using legacy ETL infrastructure:**
   Legacy ETL tools are batch oriented non-intrusive tools. In the RTDW, the needs are different, the source systems must be a part of the process. They need to inform or send data to the real-time ETL tool as the data is being entered in the Online Transactional Processing (OLTP) systems, thereby changing the paradigm from file oriented to stream oriented processing of data.

4. **Too much summary data:**
   Because of the high frequency of data acquisition in a real-time data warehouse, summary tables might very well need to be rebuilt with every change to the RTDW. To avoid this, it is advised to use materialized views which provide automatic maintenance of summary data.

5. **Lack of high availability:**
   The advantage of a RTDW is to be able to make tactical decisions based upon real-time data. Alas, too much focus is often put on the data acquisition in order to obtain real-time data. Ensuring that the

processed data is available for the knowledge workers is equally important, or the ability to make relevant and timely tactical decisions is lost.

6. **Failure to initiate business process changes:**
   Failure to change business processes within a company to ensure that real-time information is delivered and available where needed, renders the RTDW valueless.

7. **Separate ODS deployment per channel:**
   The urge to implement separate data marts or Operational Data Store (ODS) repositories in the organization to support decision making on different levels, separates the strategic and tactical decision making. This separation introduces redundant data, moving, transforming, and loading data multiple times, as well as more systems to maintain and should be avoided.

8. **Underestimating the importance of historical data:**
   Feeding a RTDW with data, keeping it up-to-date, and periodically dumping the data into a historical data warehouse, thereby separating the present from the past, relates to the previous mistake and results in less-than-optimal tactical decision making.

9. **Failure to integrate the data:**
   To avoid this mistake, focus must be put on more than just real-time data acquisition and access. An essential part of a RTDW is data integration. This means that all core organization data must be integrated into the data warehouse to provide any real value to the company.

10. **Assume all knowledge workers want real-time data:**
    Sometimes stable data is desirable over frequently changing near real-time data. E.g. a knowledge worker could be performing a strategic analysis for a business. If the data were to change just a single time during the analysis, the entire analysis could be ruined. Different SLAs for various data freshness requirements should be defined for the different knowledge workers. To provide data with different degrees of data freshness without having to replicate data, timestamps on rows in the warehouse can be used with views in order to filter out data newer than what is defined by the SLA.

2 Preliminaries

To gain a better understanding of how we can implement a real time data warehouse solution, we will re-investigate the BLIP case study, which was also addressed in the pre-master thesis. However, this time we will examine how BLIP handles live data, i.e. non-historic data. Additionally, this section will discuss existing related work and how we can utilize the results presented in the development of our RTDW system.

2.1 BLIP Case Study

This section will cover how BLIP collects tracking data and how the physical layout affects the data. Additionally, a brief description of the data manipulation performed by BLIP before it is stored in a MySQL database will be given, which requires that a couple of issues be addressed. The BLIP dataset consists of 73,986,502 tracking records within 55 active access points over 360,639 unique devices.

2.1.1 Physical Bluetooth Access Point Layout

The BLIP access point configuration does not provide full coverage of the area, which means our solution will have to handle objects that not are tracked for a period of time.

The airport consists of multiple floors, hence the access points are placed on multiple floors. A result hereof, is that an object can be tracked by two access points that are actually positioned at two different floors. This provides tracking records that are not desirable as they show movement that are not physically possible.

2.1.2 BLIP ETL

The data readings made by the BLIP tracking application can be seen as a stream of tuples of the form ($Bluetooth_{Identifier}$, $AccessPoint_{Identifier}$, $Time$), where $Bluetooth_{Identifier}$ is the unique MAC address of a Bluetooth device, $AccessPoint_{Identifier}$ is the unique MAC address of the access point that performed the reading and $Time$ is the time when the reading took place. Each reading is generated by the access points scanning for Bluetooth devices at a fixed time interval - in BLIPs case once per second. The data gathering tool made by BLIP does not support that a bluetooth device is within the proximity of two or more access points at a single point in time. This can result in readings switching very rapidly from one access point to another. This feature deviates from how related work address positioning of objects in an in-door environment.

Table 1 shows how a sample set of data generated by a number of access points could look like. T_7 is considered the current time.

Access Point 1	Access Point 2	Access Point 3
$< BT_1, AP_1, T_1 >$	$< BT_2, AP_2, T_1 >$	
$< BT_1, AP_1, T_2 >$	$< BT_2, AP_2, T_2 >$	
$< BT_1, AP_1, T_3 >$	$< BT_2, AP_2, T_3 >$	
		$< BT_2, AP_3, T_4 >$
$< BT_1, AP_1, T_5 >$	$< BT_2, AP_2, T_5 >$	
$< BT_1, AP_1, T_6 >$		$< BT_2, AP_3, T_6 >$
$< BT_1, AP_1, T_7 >$		$< BT_2, AP_2, T_7 >$

Table 1: Raw Bluetooth data.

In order to reduce the amount of raw data, data cleansing should be performed. The output format after BLIP performs their data cleansing is a tuple of the form: ($Bluetooth_{Identifier}$, $AccessPoint_{Identifier}$, *EnterTime*, *LeaveTime*), where *EnterTime* is the time when the Bluetooth device enters the proximity of an access point and *LeaveTime* is the time when the Bluetooth device leaves the proximity of an access point. This allows them to merge consecutive records for the same Bluetooth device registered within the proximity of the same access point. Table 2 shows the same data as Table 1 after performing cleansing.

$< BT_1, AP_1, T_1, T_3 >$
$< BT_2, AP_2, T_1, T_3 >$
$< BT_2, AP_3, T_4, T_4 >$
$< BT_1, AP_1, T_5, T_? >$
$< BT_2, AP_2, T_5, T_5 >$
$< BT_2, AP_3, T_6, T_? >$

Table 2: Cleansed Bluetooth data.

The amount of tuples has been decreased from 13 to 6 by cleansing the data. As T_7 is considered the current time it is possible to differ between two sets of tuples, namely historical data and live data. The records that have both an *EnterTime* and a *LeaveTime* are considered historical data, whereas the records that have an *EnterTime* but does not have a *LeaveTime* are considered live data as it reflects the current state of the tracked objects. Table 2 shows the live data by having records that has a $T_?$ as *LeaveTime* meaning that the device is still being tracked.

Table 2 shows some of the problems with the dataset, namely when a phone loses connectivity, as seen with BT_1, to an access point for a short duration of time and when a device bounces between two access points, as seen with BT_2. How this will be addressed will be discussed in Section 4.3.5.

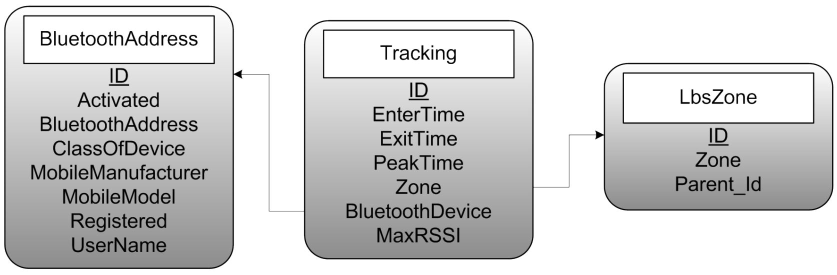


Figure 1: BLIP source schema.

After the data cleansing, BLIP stores the data in a MySQL database. The table configuration of the database is based on the data schema seen in Figure 1. The following is quoted from our pre-master thesis [11] and describes BLIPs data schema.

> The data schema consists of a main table, *Tracking*, and two supporting tables, namely *LbsZone* and *BluetoothAddress*. The *Tracking* table contains *EnterTime*, *ExitTime*, *PeakTime*, and *MaxRSSI* as well as two foreign keys to the supporting tables. *BluetoothAddress* includes more data than the name indicates, as it also contains miscellaneous data retrieved by the access points. BLIP has prepared to include additional personal meta-data through the *Registered* and *UserName* attributes. However, BLIP is not using these values in the data set. The name of a zone is stored in the *Zone* attribute in the *LbsZone* table. The attribute *Parent_Id* is used to describe a parent-child relationship between zones, which is used to define zones as a tree-structure with arbitrary height.

2.1.3 Bounce Problem

This section introduces the before mentioned bounce problem and presents an example of when it can occur. The following is quoted from our pre-master thesis [11] and describes the bouncing problem.

> The data gathering application used for the BLIP data set is not designed to handle devices that are tracked in more than one location at a time. This means that when a device is located in an area that is covered by more than one access point, the data gathering application creates *bounce records*. An example of this is shown in Figure 2 where the grey areas are overlapping areas. If a device is traversing through *AP1*, *AP2*, *AP3*, and *AP4* as indicated by the red line, there is a large possibility that the system will generate bounce records when the device is located in the grey areas.

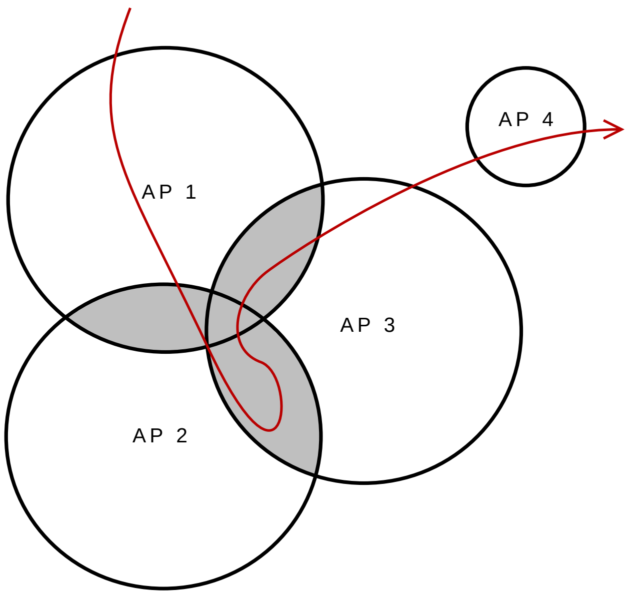


Figure 2: Bounce problem scenario [11].

2.2 Related Work

Creating a theoretical, optimal physical Bluetooth access point layout is, in theory, a rather simple task. If the area covered by a Bluetooth access point is considered as a circle, then Kershner [15] presents a model for the minimum required number of access points to provide full coverage within the area. Creating an access point setup based on Kershner's concept is, however, not valid for a number of reasons. Firstly, the sites consist of a number of obstacles that reduce the range and stability of the readers. Second, BLIP does not desire to provide coverage of a given site as they wish to focus on certain points of interest, such as check-ins, and does not wish to provide coverage of other areas, such as remote hallways.

A number of articles [1] [12] [6] cover the subject of making an optimal coverage within a closed indoor area through a number of either Bluetooth access points or RFID readers. As a common denominator, these articles assume that the entire area is covered through readers/access points and addresses positioning/moving within this area. Our solution has taken into consideration whether or not a tracked object has disappeared for certain periods of time. The objects can either disappear due to bad coverage within an access point, if it moves in an area that is not covered by the access point layout created by BLIP or simple if the device is switched off. The system we propose is able to address this problem, within reasonable time.

[14] presents the idea of building a graph on top of a RFID configuration which determines whether or not it is possible for a tracked object to move from a certain access point to another. This graph logic could be implemented in the CTF and discard invalid moves between areas covered by various access points to reduce the workload.

3 Concepts and Considerations

This section will introduce various concepts and considerations that will be utilized and taken into consideration throughout the remainder of this report. Firstly, it will introduce a definition of what a RTDW is and how it differs from a traditional DW. Secondly, it will discuss the impact of the degree of normalization on the system. Three different storage models are introduced and the benefits and drawbacks of these in regards to real-time data warehousing are discussed. Lastly, it examines the difference between how data is exported, altered and imported when using either a DW or a RTDW.

3.1 Real-Time Data Warehouse

As the term suggests, a real-time data warehouse is a system which reflects all changes in its source, or sources, in real-time or near real-time. As simple as it sounds, this is still an area of active research in the field.

Working with real time compared to historical data presents a wide range of challenges for the people designing and implementing the system, but also for the people using the end systems. Since the data available in a traditional data warehouse is not always up to date, a DW is typically used for analysis and long term improvements to overall business strategies and tactics. No changes to data through user actions are expected, or designed. The only input comes from the ETL feed as stipulated in the service level agreements. As a result, traditional DWs can be considered read-only and therefore somewhat static in nature.

A RTDW is different, as changes are reflected to the systems in (near)real-time. The operational systems are designed to accept inputs or changes to data regularly, hence have a good chance of being regularly updated. Knowledge workers can, within the scope of right-time, analyze and take action if something needs to be addressed. These actions can either be in the form of changing data values in the RTDW or being a psychical action such as redirecting traffic to avoid congestion. Performing actions does not have to be triggered by a knowledge worker. Various software implementations can take action if certain events happen. Alternatively, a Bayesian network can be trained to register and detect events by itself and take action accordingly [10]. This gives an entirely different approach toward data warehousing as the warehouse itself becomes dynamic. This can help businesses to optimize tactical business decisions and still be able to perform strategic decisions as usual.

3.2 Normalization Versus Denormalization

This section will analyze the advantages and disadvantages of using normalization on a data warehouse in regard of disk space required and query speed, and will work as a guideline for when we create the dimension(s) and fact table(s) later in this thesis. This section is based on [23] [5].

Database normalization can essentially be defined as the practice of optimizing table structures. Optimization is accomplished by analyzing the data that will be stored within the database, in particular concentrating upon how this data is interrelated. A wide range of techniques exist for normalizing a database, which are more or less strict in regard to what kind of data are allowed in different tables. All of them try to minimize the amount of redundant data stored throughout the database. Ideally, every piece of information is stored only once in the database, and all places where this information is required, it is loaded through a number of look ups throughout the tables. The most common normalization is third normal form. A normalized relational database, imposes a high read to write ratio over physical storage of data even if it is well tuned for high performance.

A normalized design often stores related, but different, information in separate logical tables. If these relations are stored physically as separate disk files, completing a database query that draws information from several tables can be slow. There are, in general, two different strategies for dealing with this. The preferred method is to keep the logical design normalized, but allow the DBMS to store additional redundant information on disk to optimize query response. In this case it is the DBMS software's responsibility to ensure that any redundant copies are kept consistent. This method is often implemented in SQL as indexed views (Microsoft SQL Server) or materialized views (Oracle). A view represents information in a format convenient for querying, and the index ensures that queries against the view are optimized automatically by the DBMS.

Denormalization is the opposite of normalization, as the name suggests. Often, queries to a RTDW require fast retrieval of the data stored. Sometimes, to accomplish the query response time needed, the decision is made to denormalize the schema. Denormalization is the process of putting one piece of information in numerous places, in order to optimize the performance of a database by adding redundant data or by grouping data. This speeds up data retrieval at the expense of data maintenance. It is the database designer's responsibility to ensure that the denormalized database does not become inconsistent. This can be done by creating constraints in the database through triggers and stored procedures, that specify how redundant data is kept synchronized.

3.3 Storage Models

There are three commonly used storage models available for storing the cubes of a data warehouse, making them available for online analytical processing. They each have different features that renders one better than the other, depending on the business model and needs of the data warehouse. Based on [7] [19] [16] [9], the three storage models are:

1. **Relational Online Analytical Processing:**
   ROLAP cubes have their data stored in a Relational Database Management System (RDBMS) along with the dimensions and hierarchies. This minimizes the amount of used storage space. ROLAP uses Just-In-Time (JIT) aggregate calculation - i.e. that aggregations are calculated on the fly when they are needed. Given the nature of the RDBMS, inserts can be performed very quickly, while calculating the aggregations can be a real time consumer. The reason for the poor query performance of the ROLAP cube is mainly the costly table joins and row by row calculation of aggregates. To remedy this, two key ideas should be kept in mind while designing the DW. First, *denormalization* - having fewer tables reduces the number of expensive joins. Second, *redundancy* - using views to store aggregates of the most common OLAP queries massively increases the response time. The greatest advantage of the ROLAP cube is the flexibility to answer queries to any level of detail desired.

2. **Multi-Dimensional Online Analytical Processing:**
   MOLAP cubes consumes much more storage space than ROLAP cubes due to the fact that MOLAP cubes consist of the fact data as well as a number of pre-calculated aggregations. The level of detail of the aggregations that are pre-calculated can be customized in order to accommodate the most frequent queries. This allows for the fastest query response time of all the OLAP cubes. The more detailed aggregations and the more dimensions that are defined, the more storage space the cube consumes. MOLAP cubes allow for offline analysis since the dimensions, facts and aggregations are stored in the cube. Unlike the ROLAP cube, MOLAP cubes do not offer the flexibility of answering queries of any level of detail. MOLAP cubes are limited to the level of detail defined by the dimensions and hierarchies in the cube. This, along with the high storage consumption, poses the greatest disadvantage of the MOLAP cube.

3. **Hybrid Online Analytical Processing:**
   HOLAP cubes store data using a combination of the two previous storage models. Aggregations are stored in a persistent cube, while detail level data are stored in the RDBMS. This minimizes the storage space, as well as optimizing query performance. HOLAP comes with the ad-

vantages of fast query performance, low storage space consumption, and flexible level of detail.

3.4 The Data Path

The need for data being processed at a quicker pace in a RTDW than a normal DW enforces changes to the way data from the OLTPs are being processed. The traditional bulk loading at fixed nightly or weekly intervals does not support the need for real-time or near real-time data. Instead, an approach where data is continuously being fed to a software system serves the same purpose of an ETL. The following sections will describe the differences between a traditional ETL and a CTF system that is used in a RTDW. The primary focus will be on how the CTF works, whereas the pre-master thesis presented a more thorough description of the ETL.

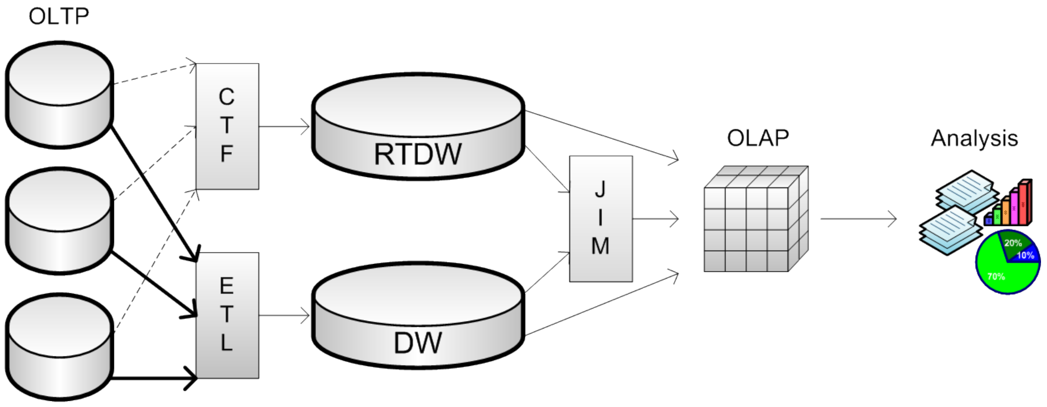


Figure 3: The RTDW and DW data path.

The similarities and differences in the data path of respectively a RTDW and a DW are shown in Figure 3. The figure will be used in the following sections to describe the data path in more detail. The figure shows how, in general, ETL and CTF both follow a similar principle in regards to processing data - namely:

- Importing data from outside source(s) - the OLTP systems.
- Performing calculations/alterations to the data set imported to fit operational needs.
- Exporting it into the end target(s), e.g. a database, text file or data warehouse.

3.4.1 Data Import

The first part of an ETL process, Extract, involves extracting the data from the source systems by bulk loading. The amount of data extracted varies from project to project, however, as it is done rather infrequent, the amount of data is typically large, as presented by the thicker lines in the Figure 3. Alternatively, the first part of the CTF: Change Data Capture (CDC) continuously feeds data, as represented by the dotted lines, from their data sources.

Change data capture is a set of software design patterns used to determine (and track) the data that has changed so that action can be taken using the changed data. Also, CDC is an approach to data integration that is based on the identification, capture and delivery of the changes made to operational data sources. There are several techniques to do this. In essence, CTF tools either push or pull data on an event driven or polling basis.

Push integration is initiated at the source for each subscribed target. This means that as changes occur, they are captured and pushed across to each target. Pull integration is initiated at the target by each subscribed target. In other words, the target system extracts the captured changes and pulls them down to the local database. Push integration is more efficient as it can better manage system resources. As the number of targets increases, pull integration becomes resource draining on the source system.

Event driven integration is a technique that involves events at the source initiating capture and transmission of changes, typically implemented through the observer design pattern. Polling involves a monitoring process that polls the status to initiate capture and application of database changes. Event driven integration conserves system resources as integration only occurs after preset events whereas polling requires continuous resource utilization by a monitoring utility.

3.4.2 Altering Data

The Altering Data stage applies a series of rules or functions to the extracted data from the source, before loading the data into the end target. Some data sources will require very little or even no manipulation of data, where others require heavy data manipulation. This stage is referred to as the Transform stage in both CTF and ETL. The following itemization presents some of the simpler rules that can be applied to a data set in this stage of the data flow:

- Encoding free-form values e.g., mapping "1" to "Male" and "M" to "Mr."
- Deriving new calculated values - e.g., creating a *sales_amount* from quantity and unit price.
- Pivoting or un-pivoting - e.g. converting multiple rows into multiple columns or vice versa.

However, more complex rules can also be applied to the imported data. An obvious example of this is the bounce problem, where a large number of data manipulations are required. More complex data manipulations are typically used in DW, where the processing time is not that important. Often, these can take historical data into consideration as well, which means that the processing time for each data record, can be long for a traditional DW. For a RTDW, the approach for this stage is different. The computation is considered important according to the *right-time* concept. As a result hereof, estimations are often made and a lightweight implementation of the transformer is implemented. The challenge with a RTDW is to propose an algorithm with as high an accuracy as possible, within the scope of right-time.

3.4.3 Data Export

The data export phase loads the entire dataset, produced by the transformer, into the end target, which is typically a database. Depending on the requirements of the organization, this process varies widely. Flow, in CTF, refers to replenishing the feed of transformed data in real-time from one or more operational systems to one or more subscriber systems. Whether a data warehouse or several data marts, the flow process is a smooth, continuous stream of information, as opposed to the batch loading of data performed by the Load stage in ETL tools.

In the case of both RTDWs and DWs, data is inserted into the database designated for the warehouse. The OLAP systems then refresh the dimensions, facts and aggregations as needed to present an up-to-date cube for browsing. Figure 3 shows how the concept of HOLAP can be viewed as the Just-In-Time Merge (JIM) between the static historical data and snapshots of the real-time up-to-date data. As described in Section 3.3, the OLAP cubes have to be reprocessed at various intervals according to the business case and storage model used. With a traditional DW, it is normally done after the execution of the ETL. With a RTDW, the data flow is constant, which means that the aggregations, facts and dimensions must be reprocessed whenever changes occur - or at specific time intervals as stipulated by the SLA.

4 System Realization

While Section 3 provides the basics and the knowledge required to build a RTDW, this section covers the actual design and implementation of the RTDW. Four main areas will be covered, where three of them are related to creating the RTDW: Describing the overall system architecture, designing and implementing the data warehouse, and designing and implementing the CTF. The last area will cover how we simulate a real-time data feed to the system.

4.1 System Architecture

The previous sections have given us knowledge and understanding of the different concepts and guide lines for designing and implementing real-time data warehouses. Now, we will put that knowledge to use by designing a real-time data warehouse that suits the business needs of the BLIP case. For brevity, we only depict and refer to data collected by BLIP that we will need in our RTDW.

While different business needs put forth different requirements for the frequency of data freshness, we consider it important to continuously maintain the RTDW with the most up-to-date data as possible, but still with a high level of data quality, i.e. data is passed to the RTDW by the flow module as soon as bounce detection and elimination has been performed. We consider the time it takes the data to be captured, transformed and fed to the warehouse to be in *right-time*, i.e. data is not real-time, it has been withheld long enough to perform data cleansing to ensure quality, but is still *fresh* enough to be regarded as the current state of the environment. However, due to the nature of the bouncing records in the BLIP dataset we cannot ensure that the most recently inserted data is valid. A threshold, called $Bounce_{threshold}$, is used to define the minimum period of time a device must be tracked within a location for it not to be considered a bouncing device. This means that the data has to be older than the $Bounce_{threshold}$ for us to ensure that the data is valid.

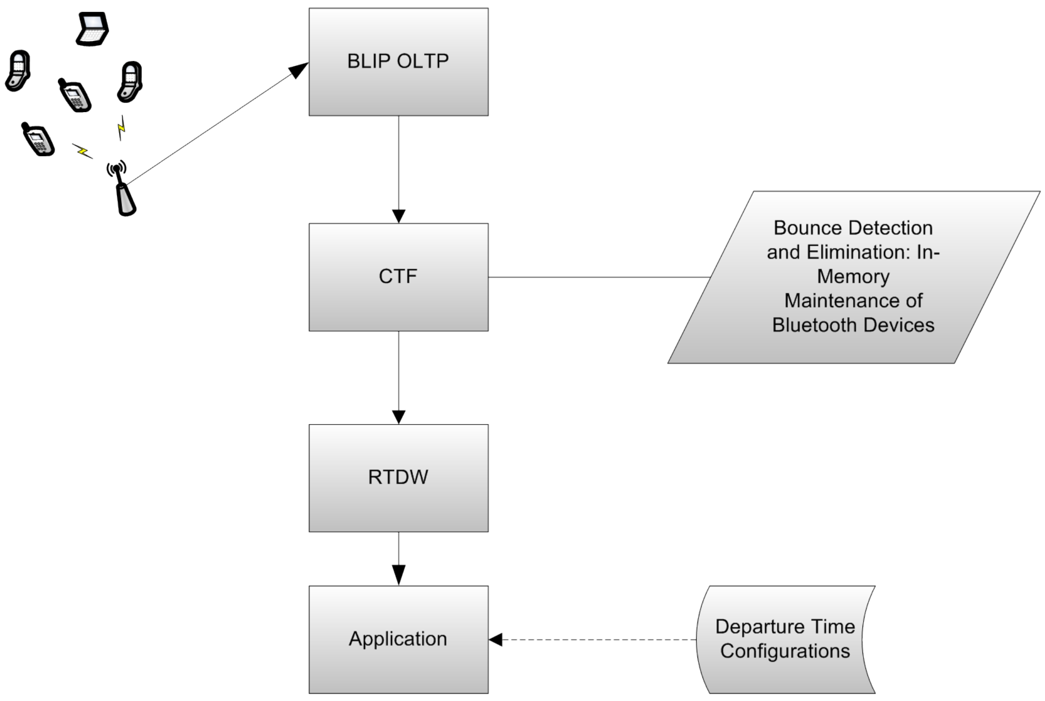

BLIP OLTP
CTF
Bounce Detection and Elimination: In-Memory Maintenance of Bluetooth Devices
RTDW
Application
Departure Time Configurations

Figure 4: System architecture.

The architecture of the system we propose is seen in Figure 4. As mentioned in Section 2.1.2, BLIP collects the Bluetooth address, access point information, and time registered for BT devices within a BLIP zone and enters it in a database. At this point in BLIPs OLTP system, our CTF will intercept the data for processing. The idea is to make the CTF scalable such that it can perform in both real- and right-time. A scalable CTF ensures that no re-engineering is needed if business needs are to change from right-time to real-time or vice versa. An in-memory maintenance of the Bluetooth devices will be used to perform on-the-fly bounce detection and elimination and transform the data. This will be elaborated in Section 4.3.5. The next step is feeding the data to the real-time data warehouse. The RTDW will be described in detail in Section 4.2. With data being fed to the real-time data warehouse, our application can query the RTDW for information that is needed, e.g. *what is the congestion level in the security area right now?*

In order to demonstrate the full potential of the RTDW, we simulate that we have access to a data mart containing the departure gate and time for all passengers. We will use this information to build an application that monitors the passengers and try to predict whether a passenger can make it to the departure gate before the given time of departure or not. The application can be implemented with various degree of detail. It can range from having a simple implementation with fixed travel times between all access points to a more complex implementation that takes into account the current congestion level in the airport as well as historic travel times between the passengers location and destination to determine an estimated travel time for the passenger. This will enable us to answer Question 3 mentioned in Section 1.1.

4.1.1 Risk Mitigation

Risk mitigation is a technique to identify and assess problem areas, and future events, that may risk the success of a data warehouse project. This technique also helps to define preventive measures to reduce the probability of these factors from occurring. We highlight three of the problem areas that are well known to cause problems when creating a RTDW [3] [17].

- *User Heterogeneity.*
  Because RTDWs are designed to address concrete business problems or opportunities, the requirements for the data warehouse are put forth by the users of the system. As a wide range of people use the systems, their needs often cover a wide range of posed questions that need answers. Data warehouse systems have to cope with a wide heterogeneous range of users. The data warehouse requirements has to address the needs of these users. While this is a problem for larger RTDWs, this does not pose a significant risk for the BLIP case, as there are a very limited range of users to the system and these users all utilize the system to extract the same type of information.

- *Growth and Scalability.*
  A user-driven data warehouse system typically grows fast, both in terms of users and data volume. Even though the amount of passengers tracked in the airport was shown to be fairly stable in the pre-master thesis, it is still desirable to implement a module-based, scalable solution. This includes both the database systems and the CTF system that performs the transformations. Some solutions even use estimations, rather than accurate calculations, to reduce workload throughout the system, which could be a useful possibility in the BLIP case. Furthermore, implementing solutions that support execution of different modules of the program in parallel should be done if possible. The

greatest risk, in regards to scalability, is the running time of the transformer. If the number of passengers being tracked were to increase, the transformer could turn out to be a potential bottleneck. How we will try to reduce the running time will be discussed in Section 4.3.3.

- *System Evolution.*
  Data warehouses are high-maintenance systems. A lot of different issues can affect the data warehouse such as changes to the products, new customer meta data, changes in production systems and similar. Therefore, the data warehouse system has to evolve with the BI changes. While system evolution poses great challenges to large organization wide RTDWs, the RTDW built for the BLIP case is limited. Should BLIP decide to include more airport logic into the warehouse, a complete redesign of the warehouse would be required.

4.2 Real-Time Data Warehouse Design

This section covers the design of the dimensions, hierarchies, and fact tables of the real-time data warehouse. First, it proposes two different designs, whose degrees of flexibility as well as space consumption are quite different. Based on a discussion, presenting the benefits and drawbacks of the two approaches, a design is finally chosen. This section will not cover the creation of keys for the various fact tables and dimensions, as we utilize the same principles for creating keys, surrogate keys and similar as described in the pre-master thesis.

As some of the dimensions are shared between the two solutions, these will be introduced before going into detail with the solutions themselves.

4.2.1 BluetoothDevice Dimension

The *BluetoothDevice* dimension, as shown in Figure 5, is used to model the Bluetooth devices tracked by the system. The dimension merely consists of the *BluetoothAddress* which is used to describe a specific device.

The additional data which are stored in the BLIP OLTP does not bring any value to the RTDW at this point and is therefore discarded. However, having implemented a dimension for Bluetooth devices enables easy expansion if the BI questions were to include additional information about the specific Bluetooth units or geographic and demographic user data.

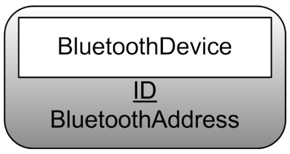


Figure 5: The *BluetoothDevice* dimension.

4.2.2 Time Dimensions

The two dimensions seen in Figure 6 are used in combination to model time. We have chosen to split time up in date and time, as we did in the pre-master thesis, to reduce the amount of records. If date and time were modeled in one table there would be over 86,400 records for each day. In a year that would give over 31 million records in the dimension which could result in a slow query performance. Furthermore, queries are often made on either a date or time basis.

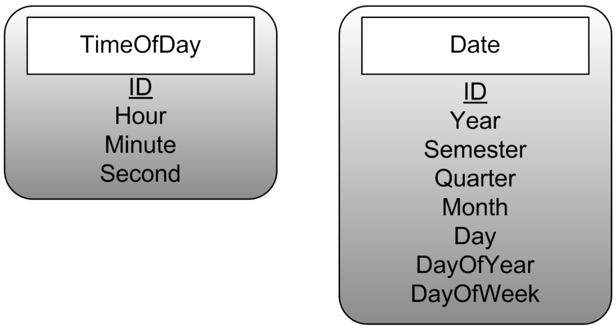


Figure 6: The *TimeOfDay* and *Date* dimensions.

The *Date* dimension consists of three attributes needed to describe a date hierarchy: *Year*, *Month*, and *Day*. To make it possible to implement a more detailed gradation the dimension includes the following attributes: *Semester*, *Quarter*, *DayOfYear*, and *DayOfWeek*.

The time dimension called *TimeOfDay* consists of the three attributes needed to specify a time hierarchy, from hour level, to a specific second: *Hour*, *Minute*, and *Second*.

4.2.3 Junk Dimension

This dimension contains the information required for the data warehouse to operate, but does not quite fit into the other dimensions. Creating a

Junk dimension to hold this data is a common practice [20] and should be considered equally as important as the other dimensions - even though the name suggests otherwise.

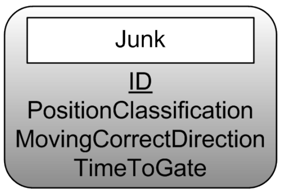


Figure 7: The *Junk* dimension.

The *Junk dimension* in Figure 7, is created to answer Question 3: *Are there any passengers that are in risk of not reaching their gate in time?*. The dimension consists of *PositionClassification*, *MovingCorrectDirection* and *TimeToGate*. The *PositionClassification* can contain three different states: `Safe`, `PossibleRisk` and `Late`. The *MovingCorrectDirection* is relevant for the passengers classified as `PossibleRisk` and it indicates whether the current movement of the tracked device is moving towards or away from the target destination. *TimeToGate* gives a rough estimate of the travel time between the current location of the device and the departure destination.

4.2.4 Specific RTDW Design Proposal

The first design we present is a very case specific solution, that can answer the BI questions proposed in Section 1, but does not provide support for answering much more than these. Figure 8 shows how the RTDW warehouse is designed.

It contains two fact tables: One for storing the tracking data and another for storing information about congestion levels. The measures that are to be stored in the *Congestion Fact* table is calculated by the CTF. While keeping track of the different objects, a counter is made for each access point. Each time a device enters or exits the proximity of an access point, its counter is either incremented or decremented. This provides a small overhead for running the CTF, however extracting information can be done by performing lookups rather than counts. Additionally, the fact table is denormalized such that it stores the access point name directly in the fact table, *Location*, rather than looking it up in a dimension. As we do not have BI questions related to different areas within the airport, but only questions about the actual locations covered by the access points, we have chosen not to include an access point hi-

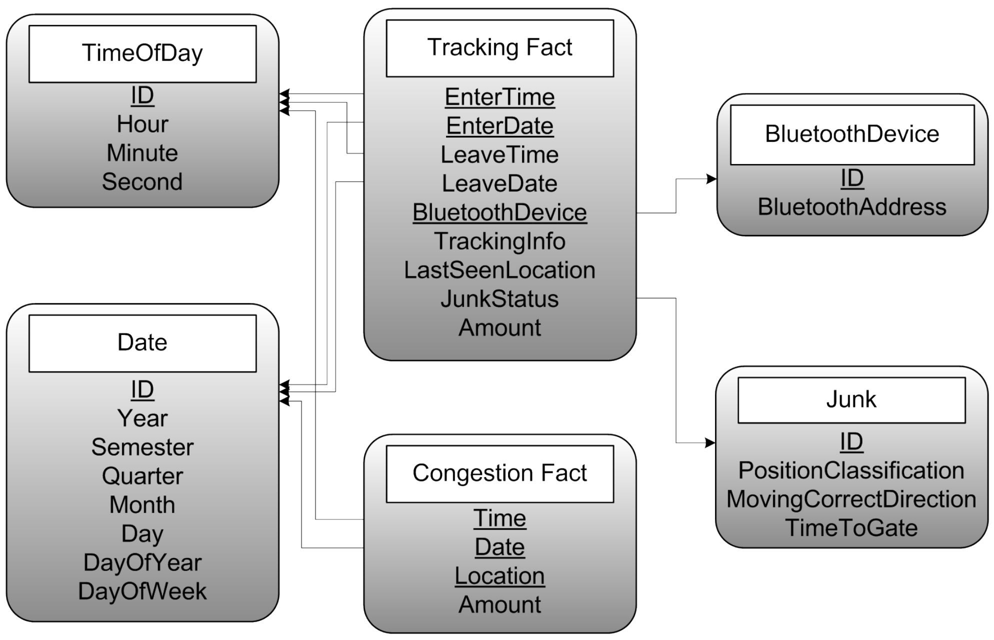


Figure 8: Specific RTDW design proposal.

erarchy. Combined, this means that performing analysis on this fact table can be performed very fast, which will ensure fast congestion analysis. However, it requires additional data to be stored in the data warehouse. The amount of records used for congestion are $Numberofaccesspoints * Numberofupdates$. Given an update frequency of one minute, with 25 access points, the result is 36.000 records pr day.

The same mindset has been used when designing the *Tracking Fact* table. The main goal when designing it was to reduce the amount of records within the tracking. The cost of reducing records is that, to some degree, certain data information details are lost. The format of the tracking data is stored in a string of the form $\{(T_{Spent_1}, AP_{ID_1}), ..., (T_{Spent_i}, AP_{ID_i})\}$ in *TrackingInfo* and is used to model movement within the airport as seen in this example:

$$\{(T_{63}, AP_1), (T_{74}, AP_2), (T_{13}, AP_3), (T_?, AP_4)\}$$

By using this data format, the presented example can be stored in one record rather than having a record for each registered tracking record. $T_?$ indicates that the record is open, i.e. the device is still being tracked.

The fact table also includes a *LastSeenLocation* fact, which can be derived from the tracking data. However, this requires the string to be loaded by the CTF and parsed before determining the position where the device was last seen. Saving the *LastSeenLocation*, as an additional fact requires little overhead as the CTF already contains the information required. However, by adding this measure it becomes easy to determine, whether or not a passenger is at risk of being late for his departure by using the *Junk* dimension without having to parse the tracking information.

Additional measures can be added if the BI questions proposed require further information, such as *Number of access points traversed in this visit?*. This is called *Amount* in the figure. However, they can only be related to the visit within the airport itself - not on each part of the traversal of the path. However, using this fact table layout presents some limitations as a lot of information is just stored implicitly. e.g. Idle- and Dwell time has to be calculated if one would require these. The introduction of this limitation is intended as it provides just enough support to BI questions posed.

4.2.5 Generic RTDW Design Proposal

The second design presented is a more generic approach and the result is a data warehouse design that is more similar to the design we proposed in the pre-master thesis. The layout can be seen in Figure 9 and it consists of one fact table and a number of dimension tables. This design is more normalized compared to the previous approach as the access points are extracted to their own dimension.

We have chosen to take inspiration from our pre-master thesis to create the *Location* dimension, however we have chosen to remove the *Site* level from the hierarchy, as we find the other levels sufficient to map the airport with a high enough degree of detail.

In order to refer to different points of interest in the airport, a hierarchy of three levels is introduced. The lowest level in the hierarchy is denoted as a *Location*, which describes the region covered by an access point. This is retrieved from the *Zone* table in the BLIP data set. A number of *Location*s form what we denote as a *Zone*. An *Area* consists of an amount of *Zone*s. The *Location* dimension is used to model the location in which a tracking record was made.

The *Tracking Fact* table contains more detailed information about the traversal of each device, as information such as idle- and dwell time can be looked

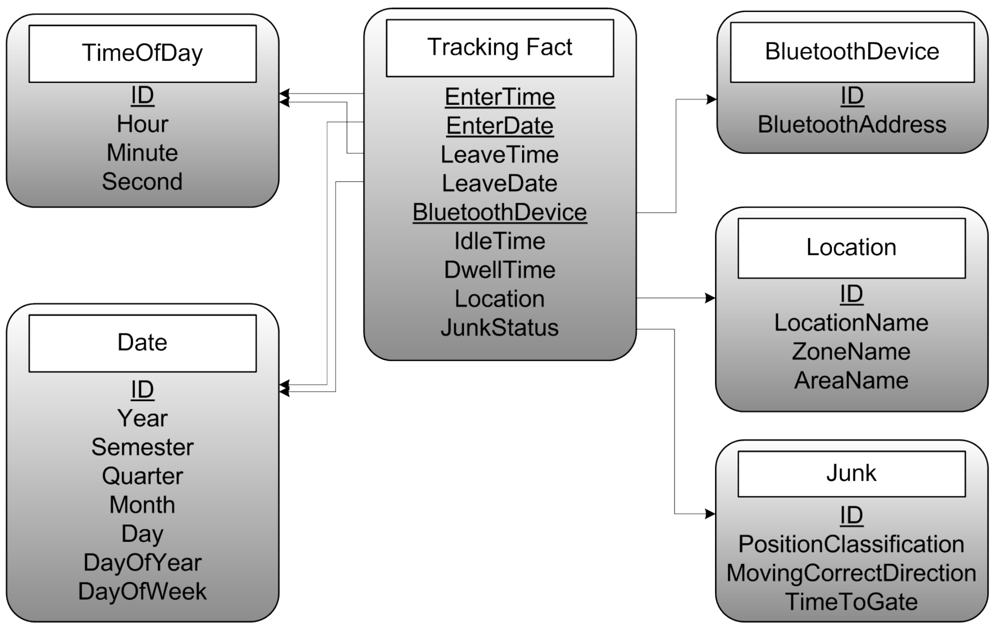


Figure 9: Generic RTDW design proposal.

up directly. Even though these are not considered important with the goals presented in Section 1, it is still a good idea to provide functionality that can answer questions that are not asked yet as described in the pitfalls in Section 1.2. Even though the second RTDW Design proposal contains one less fact table it is still possible to answer questions about congestions as live records can be examined to count congestion levels.

The main difference between the two layouts are the different fact tables. The specific design is concerned with entire visits in the airport, i.e. from the first device observation to the last, where the generic design is focused on movement between access points, i.e. it stores each movement in a separate record. Since the specific design focuses on visits within the airport, some workarounds are introduced in form of another measure, namely *LastSeenLocation* and the congestion fact table. This enables faster queries for congestion numbers as they can be looked up rather than calculated. Both designs provide same functionality in regard of determining whether a passenger is running late or not.

The respective CTF tools needed for the two systems contain small differences. The two systems handle movement from a tracked device differently. The generic system has to close a record and open a new one each time a device moves from one zone to another. The specific system, however, only performs one update on movement. It updates the *TrackingInfo* through concatenation and updates *LastSeenLocation*. Furthermore, the specific system has to close the tracking record when a device leaves an area by entering the *LeaveTime*.

As the generic design contains a higher degree of normalization, more joins are required to extract data from the tracking table. This results in a database that is faster to maintain, but is slower to query. The specific design is slightly denormalized as access point information is directly integrated into the *Tracking Fact* table. The amount of data stored in the two designs are different. The specific design reduces the amount of data within the *Tracking Fact* table, however it increases the amount of data stored for the congestion table.

The generic data warehouse design is chosen to be the implemented warehouse, as it provides support for more complex queries. While the support for more complex queries comes at the expense of slower data warehouse, we consider flexibility and expandability more important than the ability to make very specific queries fast based on Section 1.2 and 4.1.1.

4.3 Real-Time CTF

This section covers the design and implementation of the Capture, Transform, and Flow tool. It ranges from extracting the data from the OLTP to inserting data into the end target, namely the MSSQL database. It will cover the various drawbacks, benefits, limitations, and possibilities for the implementation chosen for each step of the process. Figure 10 shows a high level overview of the design for the CTF.

4.3.1 Creating a Stateful CTF

In order to optimize the data flow of the CTF, we have decided to have a singleton class, called `CTFState`, store the current state of the tool - i.e. devices and their last known and current position and time registered, as well as tracking records ready to transform and ready to insert into the RTDW. This allows the capture, transformer, and flow parts to, individually, perform optimally according to the current state. All objects in `CTFState` are *synchronized* to prevent simultaneous data access. The design contains two synchronized queues which are both stored in `CTFState` - one between

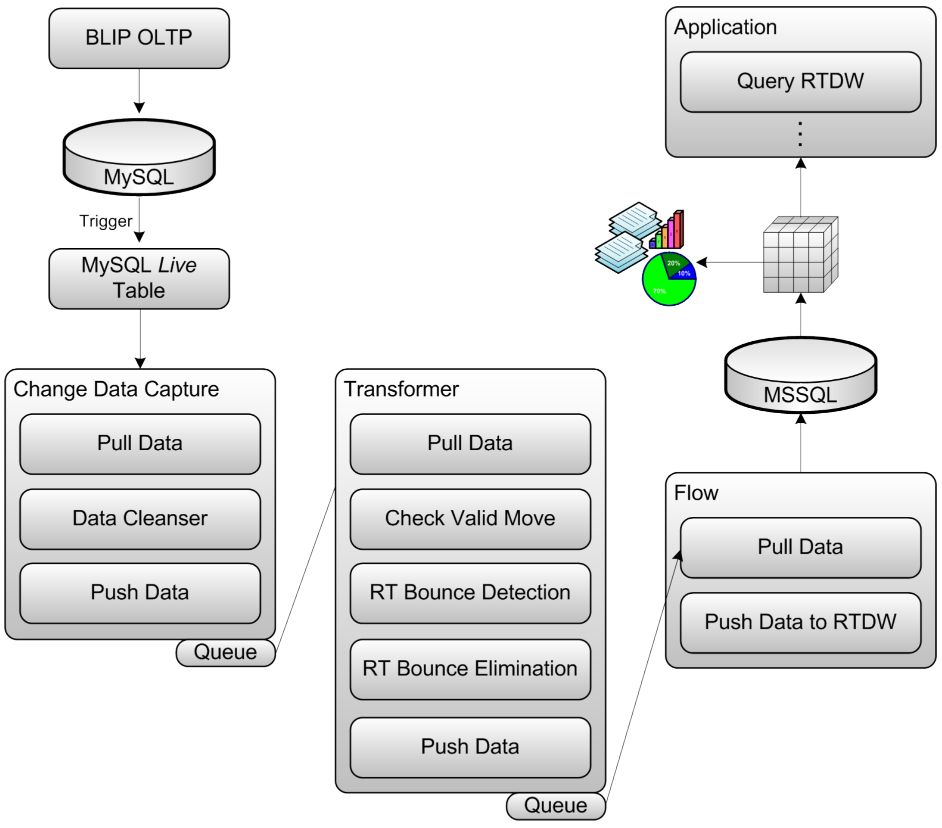


Figure 10: High level design of the CTF.

the CDC and the transformer, the other between the transformer and the flow module.

The design is quite scalable and can easily be transformed to allow for multiple threads per part. This could be accomplished by designating a queue per transformer thread and per flow thread. The thread boundaries in our case are the Bluetooth devices. Each CDC thread should perform a modulo x operation on the Bluetooth address ID - $BT_{ID}\ mod\ Num_{Threads}$ - and push the tracking record extracted on to the corresponding transformer queue. This way, load would be evenly distributed and each transformer thread would be responsible for a range of specific Bluetooth devices. In the case that there are more flow threads, a queue per thread, each handling a different range of Bluetooth devices, must be assigned in order to avoid mixing up inserts.

Our solution implements the design with a single main thread, capture thread, transform thread, and flow thread. The following goes into detail with how the CDC, transform, and flow parts work.

4.3.2 Change Data Capture

To capture the changes made in the BLIP OLTP, we first analyze how BLIP makes changes to the data source. BLIP has a main table which is populated each time either a tracking record is being created or closed, i.e. the only changes that needs to be captured comes from inserts and updates, but not deletes. The BLIP OLTP, which data source is based on a MySQL database, needs to incorporate logic to handle these changes. MySQL is limited in regards to the functionality of handling events and executing triggers. It is not possible to execute arbitrary code based on triggers or events. As a result, a more primitive approach is implemented. An insert trigger is created on the main table, which can be seen in the Listing 1.

Code example 1: After insert trigger.

```
BEGIN
  INSERT INTO Live VALUES (NEW.btAddress , NEW.
      accessPoint , NEW.enterTime , NEW.leaveTime);
END
```

The trigger seen in Figure 10, between BLIPs OLTP system and our system, replicates the inserted line into the *Live* table from BLIPs *Tracking* table. However, it does not pass information we do not consider relevant such as

MAX RSSI Strength. As a result, the schema for the table is simple as seen in Figure 11.

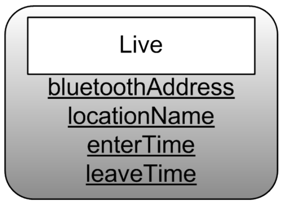


Figure 11: Schema of the live table.

The purpose of this new table is to have a have a table where the CTF can extract its data from. After extracting a line, and if its successfully pushed to the Transformer queue, that line can be removed from the `live table`. Having this extra table comes with an overhead in form of three extra SQL statements to process the data: One insert called by the trigger, one select when extracting the data to the CTF and one delete when the data is successfully pushed to the transformer. However, having this extra table ensures that the CTF can run even with a system failure. The CTF becomes state aware and knows where to resume from after a crash as long as records are processed based on ascending enter time. However, the nature of this OLTP reduces the possibility of implementing a real-time data capture. Instead, we can implement a near real-time data capture. Even if the CDC polls the database several times per second for changes, the capture will still be poll based and not event based.

A similar trigger needs to be created for handling updates, i.e. closing records. Having this update trigger enables the system to close live records and handle them accordingly.

To have actual real-time data, we would have to change the OLTP to use a different data source. One option is to use MSSQL. MSSQL enables a trigger to execute a stored procedure. A stored procedure can execute any code through the SQL Common Language Runtime(SQLCLR). Code can be written that on each insert to the MSSQL database automatically pushes information to the transformer. Another option is to use an oracle database to enable real-time data handling. Newer oracle databases bases their CDC architecture on the publisher/subscriber model, perhaps more commonly known as the observer design pattern. The publisher captures the change data and makes it available to the subscribers, this should also be the case with the announced MS SQL Server 2008 R2 - codename *Kilimanjaro* [1]. The

[1] http://www.microsoft.com/sqlserver/2008/en/us/r2.aspx

subscribers utilize the change data obtained from the publisher to perform the needed transformations before being pushed to a target site.

When extracting the data, simple cleaning is performed. The records stored in the OLTP has shown to contain corrupted data. An example hereof can be tracking records that do not contain an enter time or a record where the enter time is later than the exit time. This data cleansing is performed just before the data is pushed to the queue, which the transformer pulls from. The data cleansing could have been performed in the transformer as well, but in order to ease the workload of the transformer, it has been chosen to implement the cleansing in the CDC instead.

4.3.3 Transformer

The task of the transformer is to check all records, which are in the correct format as verified by the data cleansing module in the CDC, for valid moves and bouncing records. As Figure 10 shows, the transformer is in charge of checking the validity of the moves within the airport, perform bounce detection and elimination. To ensure scalability of the system, the transformer will be coded in a way that supports multi-threaded execution as mentioned previously.

Code example 2: Transformer algorithm.

```
WHILE true
  IF TrackingQueue is not empty AND FeedQueue is not
      full
    dequeue tracking record TR
      IF TR.BluetoothAddress exists in the state
        IF TR.AccessPoint != CurrentState.AccessPoint
          IF ValidMove(CurrentState.AccessPoint, TR.
              AccessPoint)
            IF BounceDetection(TR)
              BounceElimination(TR) - which deletes
                  bouncing record from database and
                  creates correct record
              enqueue the TR in FeedQueue - which
                  updates the new record or inserts a
                  new record
              update the state of tracked devices with
                  the bounce eliminated TR
            ELSE
```

```
12                  enqueue the TR in FeedQueue − which inserts a new record
13                  update the state of tracked devices with the new TR
14              ELSE
15                combine the CurrentState with the new state − leavetime is updated, move is discarded
16                enqueue combined TR in FeedQueue − which updates the current record
17                update the state of tracked devices with the combined TR
18            ELSE
19              combine the CurrentState with the new state − leavetime is updated
20              enqueue combined TR in FeedQueue − which updates the current record
21              update the state of tracked devices with the combined TR
22          ELSE
23            enqueue the TR in FeedQueue − which inserts a new record
24            update the state of tracked devices with the current TR
25      ELSE
26        Sleep for a while
```

The pseudocode for the Transformer is shown in Listing 2. After a tracking record is dequeued from the extracted tracking records queue, it checks whether or not the device in question has been tracked before. If not, it is directly enqueued into the feeding queue and state updates to reflect the new device as well as where and when it was last seen - as indicated in Lines 23-24. Next, we check if the access point from the new tracking record is different from the current state in Line 5. If it is not, we combine the current states enter time, with the new tracking records leave time and update the information in the state and the database, Lines 19-21. The same applies for an invalid move in Lines 15-17. Line 7 checks whether a record can be classified as a bouncing record or not, while Lines 12-13 inserts the tracking record if it is valid. Lines 8-10 performs bounce elimination on the tracking record and changes the state and the information in the database accordingly.

4.3.4 Valid Moves

One of the key components of the CTF is the sub-routine that validates whether or not the movement of a tracked object is valid. The motivation behind this routine is to, as early as possible, determine if data can be considered as invalid and thereby be discarded.

To determine whether a move is valid or not, we need a data structure to model the area. For this, we have chosen to use a graph. We consider the entire area as a graph where each vertex represents an area covered by an access point, and each edge represents a path between two areas. As this routine is to be executed each time a change is registered, it is important that the running time of the algorithm is as low as possible and that the data structure used is fast to access.

A number of data structures live up to this requirement such as hash tables and adjacency matrices. Hash tables [8] can be used in various ways giving different lookup and initialization times depending on the configuration. Perfect hashing allows for constant time lookups. This is in contrast to most chaining methods, where the time for lookup is low on average, but may be high for some sets of keys. The adjacency matrix [4] also provides constant time lookups, however it comes with a large memory overhead. Both of these data structures provide support for modeling directed-cyclic graph. This means we can implement the restriction of the movement of people enforced by the site, e.g. security, where it is not possible to go back once you have passed.

Putting forth the requirement of having a low look up time, comes with a trade off in form of higher initialization and insertion costs with both hash tables and adjacency matrices. However, due to the static nature of the access point configuration within the site, the data describing the layout is static after initialization, and does therefor not need maintenance.

For this project, it has been chosen to use an adjacency matrix, as the data structure modeling the valid moves in the site. The adjacency matrix will be built upon a directed cyclic graph library for C# called `QuickGraph`[2]. The size of the adjacency matrix would be $55 * 55$, as BLIP utilized 55 active access points for recording the data provided. As we store an `Int32` within each cell, the matrix would require 12kb of memory. A single `Bit` could also be used to indicate whether or not a pair of vertices are connected.

[2] http://quickgraph.codeplex.com

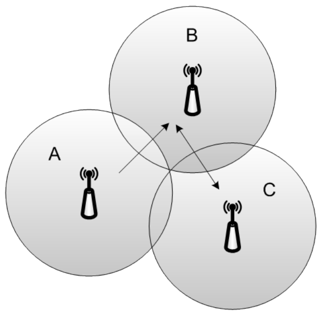


Figure 12: Example of a directed graph.

Figure 12 shows an example of how the valid moves between three access points could be and Table 3 shows the resulting adjacency matrix for the graph that can be built from the setup.

	A	B	C
A	1	1	0
B	0	1	1
C	0	1	1

Table 3: Adjency matrix for Figure 12.

The model is read as values above 0 is a valid mode, i.e. one can move from A to B, however one cannot move from B to A. The matrix also shows that it is possible for a device to move from access point A to A, which means that a device can loose connectivity to the access point and reappear later within the same access point without problems.

Listing 3 proposes an algorithm for determining whether or not a move is valid.

Code example 3: ValidMove algorithm

```
Procedure ValidMove(fromAccessPoint, toAccessPoint)
  IF AdjencyMatrix[fromAccessPoint, toAccessPoint] > 0
      RETURN true
  ELSE
      RETURN false
```

Even though the algorithm is simple in nature, it provides the CTF with a powerful tool to reduce the amount of computations needed to maintain the

RTDW. The logic for performing real-time bounce detection is only executed if a move is considered valid by the proposed model.

4.3.5 Real-Time Bounce Detection and Elimination

This section covers how we address the problem presented in Section 2.1.3 in a real-time manner. While bounce detection and bounce elimination are two separate problems, the design and implementation of these two problems are closely connected and this section will therefor cover both.

As when designing the other sub-routines, we have focused on keeping the running time as low as possible. The routine has to cover two different cases: **(i)** when a device is bouncing between two access points, **(ii)** when a device disappears and reappears within the same access point and **(iii)** when a device makes a clean move between access points. To keep the real-time bounce detection simple, we implement a solution that does not require any look ups in any external data source to detect bouncing. Rather, it should use the last live record as the reference point when determining if a move should be considered as case **(i)**, **(ii)** or **(iii)**. Case **(ii)** and **(iii)** are handled by the transformer described in Listing 2.

First, some sort of threshold have to be set for how long a device have to be tracked within an access points range for it not to be considered as bouncing, i.e. the device in question is tracked long enough to be considered a legit move. The pre-master thesis included empirical testing on a subset of the BLIP data set which showed that a $Bounce_{threshold}$ of seven seconds was optimal and will therefor be used as threshold in the examples given in this section. The bouncing problem can be addressed in several ways, of which we will discuss two solutions:

The first solution embraces the real-time concept and will insert data to the RTDW as early as possible. Inserting data early, before bounce detection and elimination can be performed, comes with the expense of possibly having invalid data in the data warehouse. Figure 13 shows an example of the presented bouncing problem.

The first record, TR_1 spans from T_1 to T_{15}, which means that the time period of the record is longer than the bounce threshold period and should therefore be correctly inserted. However, the next record moving from AP1 to AP2, TR_2, has been verified to be a valid move by the `ValidMove` algorithm, and spans from T_{15} to T_{20} which means that it is an invalid record, since the time

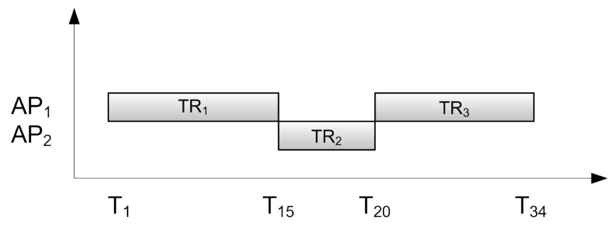


Figure 13: Example of bouncing records.

period is below the threshold. The CTF is not aware of this at time T_{15} when it inserts the record, which means it inserts a record that is actually bouncing. At time T_{20}, the CTF gets a new entry, TR_3, from the BLIP OLTP which is tracked in AP1. The CTF can then correctly identify TR_2 as a bouncing record and remove it from the data warehouse, and combine TR_1 and TR_2 into a single record spanning from T_1 to T_{34}.

Another option is to delay the insertions to the database until it has been verified that the record is not considered a bouncing record. This, however, presents the possibility to let devices disappear from the site for a short duration of time. If we use this approach on the data sample presented in Figure 13, the end result is the same. Namely, a data record from T_1 to T_{34}. The data is different in the database during the execution, however. As TR_1 has a longer duration than the bounce threshold, a record is inserted into the database at T_8 with the *enterTime* set to 1. At T_{15}, TR_1 is closed, but TR_2 is not opened until it has been confirmed that it is not a bouncing record. Since the duration of TR_2 is shorter than the bounce threshold, the client is not registered within AP_2. When the passenger re-enters AP_1, he wont be registered until it has been verified that he is not bouncing. As a result, the device is not tracked from T_{15} to T_{27}, i.e. 12 seconds where the device is not tracked.

Of these two options, we consider the first solution to be better. Even though this approach results in possible invalid data for short durations, we still consider this better than having tracked devices disappearing for shorter durations. The example in Figure 13 shows that the device is tracked at a wrong location for five seconds, whereas the other solution makes the device disappear for 12 seconds. To support this solution, we not only insert records with the flow module, but also delete and update records. This means that everything passed to the flow needs to flagged to indicate whether a record is going to be inserted, updated or deleted.

Code example 4: Real-time bounce detection.

```
Procedure BounceDetection(NewTrackingRecord TR)
  IF LastState.AccessPoint == TR.AccessPoint
    IF (CurrentState.leaveTime − CurrentState.enterTime
        ) < BOUNCE_THRESHOLD
      RETURN true
    ELSE
      RETURN false
  ELSE
    RETURN false
```

Code example 5: Real-time bounce elimination.

```
Procedure BounceElemination(NewTrackingRecord TR)

  CurrentState.dbFlag = FLAG_DELETE_NEWEST_RECORD
  FeedQueue.enqueue(CurrentState) − which deletes
     current tracking record from database

  LastState.LeaveTime = TR.LeaveTime

  LastState.dbFlag = FLAG_UPDATE_NEWEST_RECORD
  FeedQueue.enqueue(LastState)
```

Listings 4 and 5 show the logic for the bounce detection and elimination. The algorithms contain three states of the tracked device: The last seen tracking record `LastState`, the tracking record that is used in the system as the current record `CurrentState`, and the new incoming tracking record `TR`. The algorithms covers two different cases, where one has a special condition:

4.3.6 Flow

The Flow module is responsible for all communication with the MSSQL database. It handles all inserts, updates and deletes to the database depending on the status of the tracking records which is dequeued from the queue. The implementation of this module is straight forward as it, merely, sends SQL queries to database server and executes them.

5 Test

In this section we present experimental results to test two parts of the CTF, namely, to determine if the CTF produces the correct output, given a certain input, and to determine the optimal setting for the bounce threshold introduced in Section 4.3.5. We also test how, using different storage models effects the response time to the system by implementing two different cubes. Lastly, we will perform analysis of the BI related questions introduced in Section 1.1 through MDX queries and custom applications. The entire system test has been performed on a modern desktop computer, with a dual core 2.66GHz CPU and 4 GB memory, running Windows Server 2008 64bit Edition.

The data warehouse design presented in Section 4.2 has been implemented in Microsoft SQL Server 2008, the cube design has been implemented in Microsoft Analysis Services 2008 using the storage model which showed to be preferable for the BLIP case, and we have used custom applications to create the graphical output.

5.1 Real-Time CTF Test

Two tests of the CTF has been performed. One is created in order to validate that on a certain input, the correct output is generated. This test is made to validate that the proposed algorithms work as intended. Another test is made to examine the impact of the `ValidMove` and `BounceDetection` and `BounceElimination` algorithms with respect to records removed off a subset of the total dataset.

5.1.1 Capture, Transform, and Flow Correctness

To ensure the correctness of the CTF, we have extracted a small portion of the dataset which we will analyze by hand to verify that the implementation follows the rules presented in the algorithms. We have chosen a data sample from October 16th 2008, in the time period from 13:48:39 to 14:16:18. The tracked data is situated in the security area of the airport, as seen in Appendix A, and all movement is between access point 16 and 18, ID 11 and 17 respectively in the following. The figure in Appendix A is taken from [21]. The sample set consists of 22 records and can be seen in its entirety in Appendix B. The data quality of the rows are varying as some of them are only seconds long where others has a longer duration.

We will go through the three steps of the transformer, step-by-step, to check if the data matches the expected data to cover the different cases of the bounce detection. For testing the CTF, we pass the sample data through the valid move and the bounce detection/elimination algorithms. The format of the records are ($Time_{Enter}$, $Time_{Leave}$, $Bluetooth_{ID}$, $AccessPoint_{ID}$).

$$TR_1 : \text{(13:48:39, 13:48:51, 4404, 11)}$$

As this is the first record TR_1 loaded into the system the Bluetooth device has not been tracked before and are therefor stored into the RTDW without modifications. The `CTFState` singleton is updated to hold the tracking record as both the `CurrentState` and `LastState`.

$$TR_2 : \text{(13:48:51, 13:48:54, 4404, 17)}$$

The movement from access point IDs 11 to 17 is considered valid according to the adjacency matrix build upon the graph covering the area and is inserted into the database. The `CurrentState` is updated to contain information of TR_2, while the `LastState` still contains TR_1.

$$TR_3 : \text{(13:48:54, 13:49:05, 4404, 11)}$$

TR_3 is recorded within the proximity of access point 11. As the duration of TR_2 is below the bounce threshold, set to seven seconds, action has to be taken. TR_2 will be removed from the RTDW and TR_1 will have to be modified. Rather than inserting a new record for TR_3, we combine TR_1 and TR_3 to span from the $Time_{Enter}$ of TR_1 to $Time_{Leave}$ of TR_3. When the time 13:49:05 is reached, the RTDW will contain one record spanning from 13:48:39 to 13:49:05 in the access point with ID 11.

$$TR_3 : \text{(13:48:39, 13:49:05, 4404, 11)}$$

The same techniques are applied on the remaining records in the sample data set from Appendix B. The result, shown in Table 4, is that the 22 records, after performing move validity checks and bounce detection and elimination, are reduced to 8 records. Running the implementation of our CTF and checking the results yields the same results presented here. This means that the correctness of the CTF has been verified according to the design of our algorithms.

enterTime	leaveTime	btAddress	accessPoint
13:48:39	14:05:17	4404	11
14:05:35	14:06:20	4404	17
14:06:20	14:06:34	4404	11
14:06:37	14:07:59	4404	17
14:08:01	14:08:14	4404	11
14:08:20	14:14:41	4404	17
14:15:19	14:16:18	4404	11

Table 4: Sample data after bounce detection and elimination.

5.1.2 Valid Move, Bounce Detection And Elimination Test

In this experiment we load a subset of the dataset into the data warehouse using different $Bounce_{threshold}$ values, (0 - 10 seconds). We have chosen to work with a subset of 25,000 records from the BLIP dataset, due to the real-time nature of our CTF. The 25,000 extracted records ranges from 13:48:32 to 16:12:54 on October 16th 2008, which is a span of 144 minutes and tracks 1,641 unique devices within 23 access points. The results are shown in Figure 14.

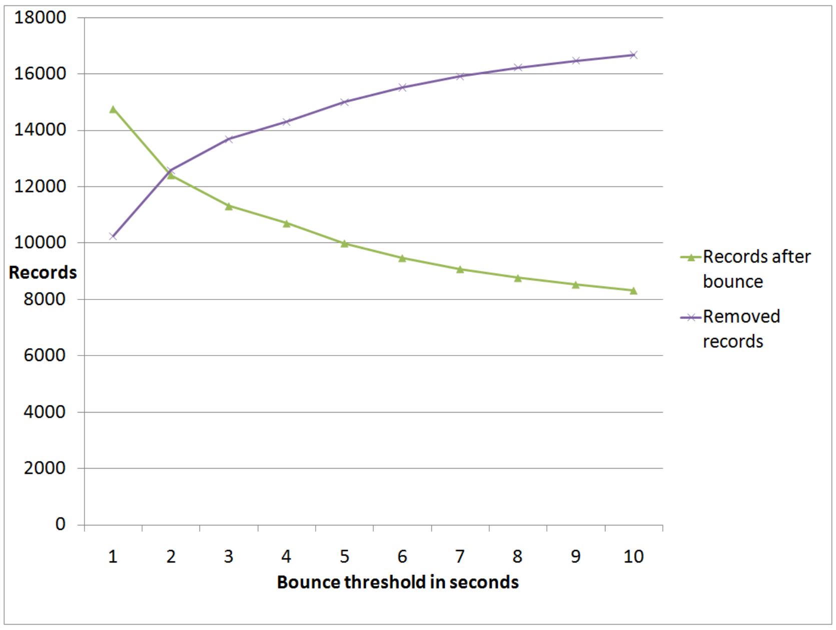


Figure 14: Effect of the $Bounce_{threshold}$ attribute on the number of records.

The results show that the amount of records decrease rapidly in the beginning and flattens out at approximately a 7 second threshold. This is consistent with the result from the pre-master thesis, indicating that a bounce threshold of 7 seconds is optimal.

If the $Bounce_{threshold}$ is set too high, records that are not actual bounce records, but merely people crossing in an access points peripheral, will be marked as bounce records. If it is set too low, too few of the bounce records are eliminated. We set the $Bounce_{threshold}$ to 7 seconds, which reduces the amount of records from 25,000 to a total of 9,082 tracking records, which is equal to a reduction of the tracking records by 63.67%. In comparison, the old bounce detection presented in the pre-master removed 2,642,095 out of 21,161,406, which equals to a reduction of 12.49%.

Running the same test on different subsets of the dataset with various starting point in time and various duration shows that the reduction in records spans from 50% to 72% with a $Bounce_{threshold}$ set to 7 seconds. This degree of data quality ensurance supports the statement of Goal 3, regarding increasing the data quality output.

5.2 Storage Model Test

To test the response time of the real time data warehouse and the impact the choice of the storage model have on the system, we have implemented the cube with two different storage models in the Microsoft SQL Server Analysis Services 2008 [18], namely ROLAP and HOLAP.

The first is a pure ROLAP cube, where all detail data, dimensions, and aggregations are stored in a relational format. The server is configured to listen for changes in data, and all queries done to the cube are therefore performed on the current state of the data, this is defined by setting the proactive caching setting to real-time ROLAP.

The other cube is implemented using the real-time HOLAP proactive caching setting and the HOLAP storage model, i.e. detail data is stored in a relational format, while the aggregations are stored in a multidimensional format. As with the ROLAP cube, the HOLAP cube listens for changes in the data source, however the changes are only reflected to the aggregations when the cube has been rebuilt. Using the real-time HOLAP introduces faster query times when querying aggregated data, but at the expense of introducing additional workload through frequent cube reconstructions.

To test the response time of the two cube designs, the Multi Dimensional Expression (MDX) queries which can be seen in Listings 6 and 7, have been created which both displays the count of tracking records of Bluetooth devices at a given location. Listing 6 is set to a *Zone* detail level in the *Location* hierarchy, whereas Listing 7 is set to the *Location* level. An MDX query is to Analysis Services, what a SQL query is to SQL Server. MDX is a query language used to manipulate and retrieve data from a cube.

Code example 6: Tracking count of devices distributed by zones.

```
SELECT NONEMPTY([Dim Location].[Location Hierarchy].[
    Zone Name]) ON COLUMNS,
  [Dim Bluetooth Device].[Bluetooth Hierarchy].[
      Bluetooth Address] ON ROWS
FROM [Cube]
WHERE [Measures].[Fact Tracking Count]
```

Code example 7: Tracking count of devices distributed by locations.

```
SELECT NONEMPTY([Dim Location].[Location Hierarchy].[
    Location Name]) ON COLUMNS,
  [Dim Bluetooth Device].[Bluetooth Hierarchy].[
      Bluetooth Address] ON ROWS
FROM [Cube]
WHERE [Measures].[Fact Tracking Count]
```

Because the HOLAP cube stores aggregations in the multidimensional format, we expect this storage model to be faster than the ROLAP model that has to look up each tracking record and group it by their zone identifier to return results on a zone level. Table 5 displays the average running time of the two MDX queries performed on the ROLAP and HOLAP cube, respectively. We can see that the aggregations stored in the multidimensional format in the HOLAP cube reduces the query response time of queries using the *Zone* level from 44.45 seconds to 33.15 seconds. Queries using the *Location* level performs similar on both the ROLAP and the HOLAP cube. This is because neither cube stores aggregates on the lowest level of detail. Custom aggregation settings can be set to also include the most detailed level, at the cost of more storage space.

Even though the HOLAP cube provides slightly better response times, we still estimate that the pure ROLAP implementation is preferable. Regardless of the state of the cube, you are guaranteed an answer where the response time is the processing time without any additional latency from re-processing

$\frac{StorageModel}{HierarchyLevel}$	ROLAP	HOLAP
Zone	44.45 sec	33.15 sec
Location	65 sec	64.45 sec

Table 5: Average query response time.

the cube. Both systems are viable for using real-time data warehouses, however, depending on the BI questions and requirements put forth in the SLA ,one might be preferable over the other.

5.3 Real-Time Data Simulation

As we do not have access to the actual data stream from the BLIP OLTP, we have created a system that simulates a continuous stream of real-time data based on the historical dataset provided by BLIP. In order to simulate this stream, we have a program that inserts tracking data into a temporary table from which the CTF captures the data. The structure of this program can be seen in Listing 8. Once the data is inserted into the temporary table, the trigger from Section 4.3.2 is executed and the data is passed on to the *LiveTable*, which the capture module streams data from.

Code example 8: Real-time tracking simulator.

```
Procedure ExtractRecords()
WHILE true
  IF SimulatorQueue is not full
    Extract valid tracking records
    Enqueue records into the SimulatorQueue
  ELSE
    Sleep for a while

Procedure InsertRecords()
Set CurrentTime to first records EnterTime
WHILE true
  IF SimulatorQueue is not empty
    Dequeue tracking record TR
    IF TR.EnterTime == CurrentTime
      Insert record to temporary table in DB - which
          simulates the stream of tracking records from
          BLIPs OLTP
    ELSE
      Set CurrentTime to TR.EnterTime
      Sleep the time difference between CurrentTime and
          EnterTime
  ELSE
    Sleep for a while
```

Using the procedures in Listing 8 we are able to simulate a continuous stream of data to the CTF in real-time. Additionally, we are able to increase the flow rate speed, in order to run the simulation in e.g. 2x speed, or basically as fast as the system is able to process the records.

5.4 Real-Time Data Warehouse Test

To demonstrate the power of the RTDW, we have implemented some sample applications. Analysis of the cube can be done through the use of software like TARGIT BI Suite and Business Intelligence Studio, however this section focuses on creating domain specific solutions, that queries the RTDW through the ADOMD.NET[3] data source - which is used to access and query multidimensional data warehouses. The first application we propose is a software solution that monitors where each tracked device is and reports if congestions exist within the site. The other is an application to check for passengers that are running late.

5.4.1 Congestion Level Application

One of the uses of the congestion level application is to present how the congestion level is within different areas of the airport. This application will answer BI Questions 1 and 2. In order to answer these questions, a C# application has been built that polls the RTDW cube and extracts the data needed. This is done by using the MDX query shown in Listing 9.

[3] http://msdn.microsoft.com/en-us/library/ms123483.aspx

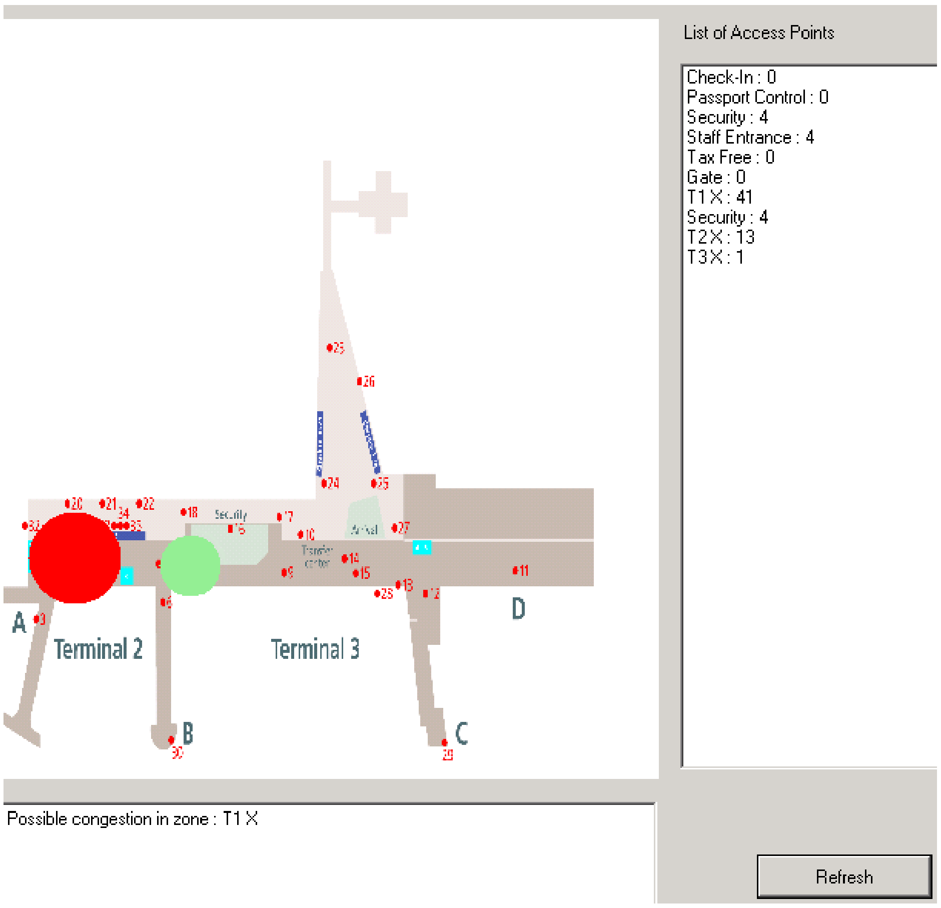
Code example 9: MDX query for congestion levels.

```
1 SELECT [Dim Location].[Location Hierarchy].[Zone Name].
    MEMBERS ON ROWS,
2   FILTER (
3     DESCENDANTS (
4       { [Enter Time].[Time Hierarchy].[Minute].&[@Hour
          ]&[@FromMinute]:
5       [Enter Time].[Time Hierarchy].[Minute].&[@Hour]&[
          @ToMinute] },
6       [Enter Time].[Time Hierarchy].[Minute] ),
7     [Measures].[Bluetooth Device Distinct Count] ) ON
        COLUMNS
8 FROM [Cube]
9 WHERE ( [Enter Date].[Date Hierarchy].[Day].&[@Year]&[
    @Month]&[@Day] )
```

Line 1 specifies that all members of the *Location* hierarchy at the *Zone* detail level are represented as rows. Lines 2-7 specifies that the dynamic timespan is represented as columns, and that the measure is the distinct count of Bluetooth devices. Line 9 limits the query to a specific date.

The main element of the application is an image showing the airport. Based on the query results, graphical elements are drawn on top of the image. These elements vary in size and color, depending on the number of people within the proximity of each access point ranging from smaller green circles, i.e. few devices are tracked within an access point, to larger red circles, i.e. many devices are tracked within an access point. In addition to the graphical elements in the application, different logic can be implemented depending on the business questions presented by BLIP. An example of this could be sending notifications to people of interest, when certain events occur. An event, in example, could be if the amount of people within an access point or a set of access points exceeds or drops below a certain threshold.

Figure 15 shows an implementation of the application for monitoring congestions using a sampled data set. The application consists of three main parts: The before mentioned image which shows the airport and the currently monitored units within the site, a listing containing the same information as the image and a list box that prints all events that has occured. For overview purposes, we have chosen to draw congestion levels on the map, only when the number of devices registered exceeds 10.



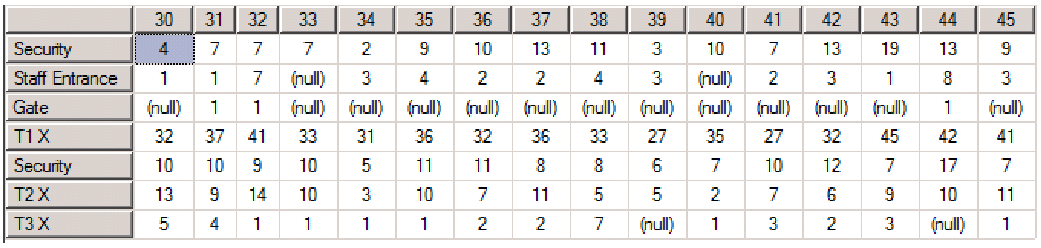
Check-In : 0
Passport Control : 0
Security : 4
Staff Entrance : 4
Tax Free : 0
Gate : 0
T1 X : 41
Security : 4
T2 X : 13
T3 X : 1

Possible congestion in zone : T1 X

Figure 15: Sample implementation of congestion monitoring application.

Figure 15 shows monitored objects within two access points and a single captured event, namely a *Possible Congestion* event. With the current implementation, the information is merely printed in the list box, however in a real world application, relevant people could be contacted through e-mail, SMS or similar.

Question 2, *Are there any congestions forming?* is closely connected to Question 1, *Are there any congested areas in the airport?*. Both operate on the measure *Bluetooth Device Distinct Count*. The main difference is that where one focuses on the most recent data the other takes historical data into consideration as well. To predict whether or not congestions are forming, one have to examine the congestion levels for each zone and how the recent growth tendency has been within the actual zone.



	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Security	4	7	7	7	2	9	10	13	11	3	10	7	13	19	13	9
Staff Entrance	1	1	7	(null)	3	4	2	2	4	3	(null)	2	3	1	8	3
Gate	(null)	1	1	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	(null)
T1 X	32	37	41	33	31	36	32	36	33	27	35	27	32	45	42	41
Security	10	10	9	10	5	11	11	8	8	6	7	10	12	7	17	7
T2 X	13	9	14	10	3	10	7	11	5	5	2	7	6	9	10	11
T3 X	5	4	1	1	1	1	2	2	7	(null)	1	3	2	3	(null)	1

Figure 16: Sample report of congestion levels.

Figure 16 shows the result of a MDX query posed through Microsoft SQL Server Management Studio, which shows the congestion levels for the last 15 minutes from a certain point of time. This result set can be used by the airport personnel to manually examine the congestion tendencies within the airport for the selected duration, or by an application to automatically predict future congestion levels.

6 Conclusion

The main focus of this report is to provide effective real-time analysis of real-time tracking data collected by BLIP at Copenhagen Airport. To accomplish this, a real-time data warehouse has been designed and implemented to address the goals and questions presented in Section 1.1.

To provide a basis for the project, we re-introduce the BLIP case. We take into account how BLIP models real-time data, through the use of open records and examine related work, that addresses the problem of tracking objects within an indoor environment, and how our contributions differ from what other articles present. Section 3 introduces various concepts and considerations that are utilized and taken into consideration in the process of designing and implementing the RTDW.

We introduce two different real-time data warehouse designs, a specific and a generic design, and choose to implement the RTDW, using the generic design, in Microsoft SQL Server 2008. We define the facts, dimensions and hierarchies required for performing an analysis of the data in the data warehouse. Additionally, we define the right-time scope for the RTDW. We consider the time it takes the data to be captured, transformed and fed to the warehouse to be in right-time, i.e. data is not real-time, it has been withheld long enough to perform data cleansing to ensure quality, but is still fresh enough to be regarded as the current state of the environment. The design is flexible by nature and is easy to maintain. This fulfills Goal 1, *Creating a fast, flexible, and durable real-time data warehouse.*

The CTF application developed in Section 4.3 captures the changed data, then transforms the data to suit the design of the data warehouse and lastly loads the data into the warehouse through the flow module. The CTF utilizes the proposed algorithms in Section 4.3.4 and 4.3.5 to perform data cleansing and reduce the amount of data inserted into the RTDW. With a $Bounce_{threshold}$ set to 7 seconds, the amount of tracking records is reduced from 25,000 to 9,082, which is equal to a reduction of tracking records by 63.67%. In comparison, the bounce detection and elimination algorithm presented in the pre-master thesis removed 2,642,095 out of 21,161,406 using the same threshold, which equals a reduction of 12.49%. If we were to apply our real-time bounce detection and elimination algorithm to the complete data set, consisting of 73,986,502 records, we would end up with removing approximately 47,107,205 records, reducing the data set to a total of 26,879,297 records. In comparison, approximately 9,240,914 bounce records would be removed, reducing the total data set to 64,745,588 records, using the algorithm from the pre-master thesis. Due to the nature of the data, data collection process, and physical access point layout, it is safe to say that the quality of the data contained within the data warehouse have been drastically increased. This fulfills Goal 2, *CTF data throughput*, and Goal 3, *CTF data output quality.*

The choice of data storage model has proven to be noticeable in the query response time. To choose the optimal storage model, a vast analysis of the types of queries to be performed, the number of users and applications accessing the data warehouse and the frequency of queries has to be conducted.

Such an analysis is beyond the scope of this thesis, however, given the rate of data flow, total number of tracking records, and types of queries we have performed, the optimal storage model would be ROLAP. This is due to the length of time a device is being tracked in general, as well as the number of tracking records created per device per day. We believe that the constant updates, combined with the constant need to reprocess the MOLAP aggregation cube, in a HOLAP storage model, would exceed the extra calculation time of querying a ROLAP cube.

We have shown how the implemented solution can be used to answer Question 1, *Are there any congested areas in the airport?*, and Question 2, *Are there any congestions forming?*, by executing MDX queries and using the results to visualize the state of the environment. To optimize the execution time of the queries, the data warehouse can be customized to support the aggregations needed to calculate the results of specific queries faster.

6.1 Contribution List

In order to clarify the contributions we have made in this report, we create a contribution list. The list contains a brief description of why we examine the subject and how the subject has been implemented.

- Valid Move Algorithm.
  As we wish to reduce the workload of the CTF and heighten the data quality, we want to reduce the amount of data, that has to be processed as early as possible. We present an algorithm for determining whether a move within the airport is considered valid or not. To determine if a move is valid or not, we created a graph over the airport, where each vertex represents an area covered by an access point, and each edge represents a path between two areas. We discuss different data structures to model the graph and choose an adjacency matrix, as it has a low worst time running speed even though it has a small memory overhead.

- BounceDetection Algorithm.
  The BLIP OLTP creates a number of bounce records each time a tracked device is seen within two access points. In order to cleanse the data provided by BLIP, and thereby increase the quality of the data we insert into the RTDW, we propose an algorithm which detects when a device is bouncing between access points. The real-time

bounce detection does not communicate with the DW to look up previous records, but only uses the state stored in the `CTFState`. The algorithm presented is more streamlined to the BLIP business case than the algorithm proposed in the pre-master thesis, and therefor provides data of a higher quality.

- BounceElemination Algorithm.
  Since the records processed by the transformer are stored into the RTDW as soon as possible, the data in the RTDW might be invalid due to bouncing records. It is not possible for us to ensure the correctness of the data in the RTDW until the data is at least older than $Bounce_{threshold}$. The *BounceElemination* algorithm is responsible for cleaning up data in the RTDW that has proven to be invalid due to bouncing records. This periodically faulty data can only occur in the newest entries in the RTDW, and does not conflict with the requirements of the business case and can therefor be justified. From a historical point of view, the data will always be static and correct.

7 Future Work

This section discusses ideas for future development. With the current stage of development, we have a RTDW that can track movement within the airport and answer BI questions of both a historical and real-time nature.

We will discuss two possible ways to continue the development of this project. Continuing the development can focus on two different areas, namely: Expanding functionality or improving the current RTDW.

1. Expanding functionality.

   As the sample application have shown, it is possible to give the users of the airport a better experience as the airport personnel can detect and take actions when the congestion levels within the airport is rising. This is made possible by having up-to-date information about the devices within the airport. A wide array of applications are made possible by having real-time information about how the people spend their time within the airport. By examining the patterns of each device in real-time it becomes possible to push relevant information to the Bluetooth device in question. If a device has been tracked for longer durations within two stores, that sell similar products, the airport can push data to the device containing information about similar products at a third store. A token can also be pushed to devices that are moving toward a certain store, which can be turned in for a discount or a free product at said store.

Should our real-time data warehouse be integrated with the various other data sources hosted by Copenhagen Airports, it is even possible to push more specific information to the various devices based on additional meta-data about the devices. If a device is owned by a passenger flying to Italy, but the check-in is not until two hours later, then the device can get a notification stating that an Italian restaurant within the airport might be interesting.

This enables the airport to give passengers with a Bluetooth device, a more personal and unique experience within the airport. It also enables the shops in the airport to do very specific advertising to just the people that are within their key demographics.

2. Improving the RTDW.

   Another way to go is to take the current RTDW solution and make further optimizations to it. Tests have shown that the bottlenecks within the CTF have shown to be when extracting data and inserting data as the live table and the queue to the flow module slowly increase in size. As a result, structural changes can be made to the hardware setup of the system to reduce the latency between the directly connected subsystems.

   Another way to reduce the work load of the flow system, is to reduce the amount of records it has to insert. One way to reduce the amount of data is to reconsider how movement is tracked within the airport. As it is with the current system, each passenger is being tracked individually. Often, people walk together in groups, this observation can be incorporated into the system. Only minor changes would be required to the data warehouse design. Each tracking record would have to contain a weight indicating the size of the group and the Bluetooth device measure would have to be able to contain a list of Bluetooth device IDs rather than a single ID.

   The transformer would have to include algorithms for being able to dynamically, and in real-time, be able to generate groupings of people and to ungroup people. Creating these groupings can be based on meta data: Departure time, flight number, departure gate and similar. Grouping can also be done by examining the movement of the various devices, or a combination that take both meta data and movement into account.

While this solution transforms a number of devices into a single group object, and thereby decreases the workload of the flow module, it also increases the workload of the transformer. It also presents the possibility to easily pose queries that only deal with groups of people rather than individuals or vice versa.

References

[1] Bahl, P., and Padmanabhan, V. N. RADAR: An In-Building RF-Based User Location and Tracking System. In INFOCOM, 2000.

[2] Brobst, S. Ten Mistakes to Avoid When Constructing a Real-Time Data Warehouse. `http://www.tdwi.org/research/display.aspx?ID=6792`, 2003, Accessed: 06/19/2009.

[3] Bruckner, R. M., List, B., and Schiefer, J. Risk-management for data warehouse systems, 2001.

[4] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.

[5] Date, C. J. *An Introduction to Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[6] Davison, A., and Murray, D. Simultaneous Localization and Map-Building Using Active Vision. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2002.

[7] Golfarelli, M., and Rizzi, S. *Data Warehouse Design - Modern Principles and Methodologies*, 1st ed. McGraw-Hill, 2009.

[8] Goodrich, M. T., and Tamassia, R. *Data Structures and Algorithms in Java*, 4th ed. John Wiley & Sons, 2001.

[9] Hackney, D. What are the Major Differences Between ROLAP and MOLAP? `http://www.information-management.com/news/1330-1.html`, 1999, Accessed: 06/19/2009.

[10] Hamm, D. C. K. Viewing adaptive bayes network results. `http://www.dba-oracle.com/data_mining/t_data_mining_bayes_network.htm`, 2008, Accessed: 12/19/2008.

[11] Hansen, J. T., Jørgensen, S., Tinggaard, S. N. M., and Wejdling, R. L. Pre-master Thesis: A Data Warehouse Solution for Analysis on Indoor Tracking Data, 2008.

[12] Hähnel, D., Burgard, W., Fox, D., Fishkin, K., and Philipose, M. Mapping and Localization with RFID Technology. In ICRA, 2006.

[13] JACOBS, B. Database Replication: Solving Real-Time Movement of OLTP Data. Business Intelligence Journal volume 13, 2008.

[14] JENSEN, C. S., LU, H., AND YANG, B. Graph Model Based Indoor Tracking. Proceedings of the Tenth International Conference on Mobile Data Management, Taipei, Taiwan, 2009.

[15] KERSHNER, R. The Number of Circles Covering a Set, 1939.

[16] LACHEV, T. *Applied Microsoft Analysis Services 2005: And Microsoft Business Intelligence Platform.* Prologika Press, 2005.

[17] LANGSETH, J. Real-Time Data Warehousing: Challenges and Solutions. `http://dssresources.com/papers/features/langseth/langseth02082004.html`, 2004, Accessed: 06/19/2009.

[18] MICROSOFT. Choosing a Standard Storage Setting. `http://msdn.microsoft.com/en-us/library/ms175646.aspx`, 2008, Accessed: 07/19/2009.

[19] N/A. Data Warehousing Tutorial. `http://www.scribd.com/doc/2066276/Data-Warehousing-Tutorial`, 2008, Accessed: 06/19/2009.

[20] ROSS, M. Kimball Design Tip #48: De-Clutter With Junk (Dimensions). `http://www.kimballgroup.com/html/designtipsPDF/DesignTips2003/KimballDT48DeClutter.pdf`, 2009, Accessed: 10/07/2009.

[21] TINGGAARD, S. N. M., AND WEJDLING, R. L. A Data Warehouse Solution for Flow Analysis Utilising Sequential Pattern Mining, 2009.

[22] Blip Systems A/S - Bluetooth Marketing Solutions. `http://www.blipsystems.com/`, 2009, Accessed: 06/04/2009.

[23] ZAKER, M. Data Warehouse Design Considerations. `http://fit.mmu.edu.my/news/newsattach/Morteza%20Zaker_abstract_.pdf`, 2007, Accessed: 07/07/2009.

[24] ZONE, S. L. A. Service Level Agreement - Information and background for SLAs. `http://www.sla-zone.co.uk/`, 2009, Accessed: 07/07/2009.

A Access Point Placements

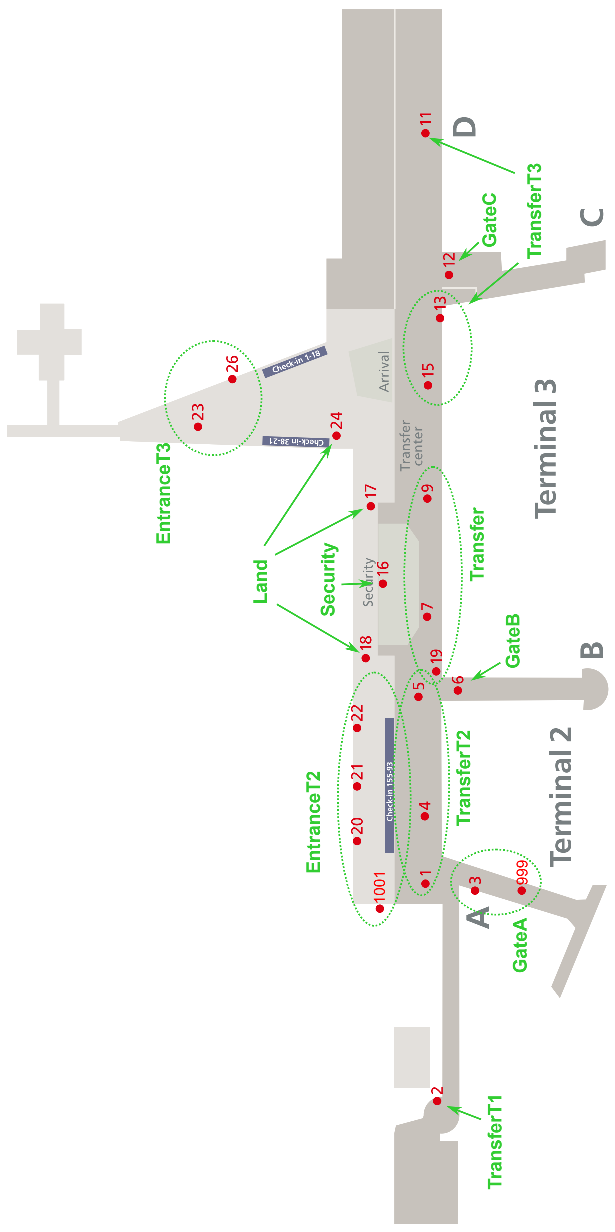


Figure 17: BLIP access point placements. The red dots represent access points, and the green text/circles represent zone divisions [21].

B Sample Dataset

Data Sample Before Bounce Detection And Elimination			
enterTime	leaveTime	btAddress	accessPoint
2008-10-16 13:48:39	2008-10-16 13:48:51	4404	11
2008-10-16 13:48:51	2008-10-16 13:48:54	4404	17
2008-10-16 13:48:54	2008-10-16 13:49:05	4404	11
2008-10-16 13:49:07	2008-10-16 13:49:07	4404	17
2008-10-16 13:49:08	2008-10-16 13:49:31	4404	11
2008-10-16 14:04:41	2008-10-16 14:05:17	4404	11
2008-10-16 14:05:35	2008-10-16 14:06:20	4404	17
2008-10-16 14:06:20	2008-10-16 14:06:34	4404	11
2008-10-16 14:06:37	2008-10-16 14:07:16	4404	17
2008-10-16 14:07:21	2008-10-16 14:07:21	4404	11
2008-10-16 14:07:24	2008-10-16 14:07:49	4404	17
2008-10-16 14:07:59	2008-10-16 14:07:59	4404	17
2008-10-16 14:07:59	2008-10-16 14:07:59	4404	11
2008-10-16 14:08:01	2008-10-16 14:08:14	4404	11
2008-10-16 14:08:20	2008-10-16 14:09:18	4404	17
2008-10-16 14:09:19	2008-10-16 14:09:19	4404	11
2008-10-16 14:11:38	2008-10-16 14:12:10	4404	17
2008-10-16 14:12:29	2008-10-16 14:12:29	4404	11
2008-10-16 14:12:50	2008-10-16 14:13:08	4404	17
2008-10-16 14:13:09	2008-10-16 14:13:09	4404	11
2008-10-16 14:13:09	2008-10-16 14:14:41	4404	17
2008-10-16 14:15:19	2008-10-16 14:16:18	4404	11

C Summary

This project concerns the design and implementation of a real-time data warehouse for analysis on tracking data collected at Copenhagen Airports A/S. The project is done in cooperation with BLIP Systems A/S, who has a Bluetooth based tracking system installed in Copenhagen Airports A/S. The BLIP dataset consists of 73,986,502 tracking records within 55 active access points over 360,639 unique devices. The data readings made by the BLIP tracking application can be seen as a stream of tuples of the form ($Bluetooth_{Identifier}$, $AccessPoint_{Identifier}$, $Time$). In order to reduce the amount of raw data, data cleansing is performed by BLIP. The output format after BLIP performs their data cleansing is a tuple of the form:

($Bluetooth_{Identifier}$, $AccessPoint_{Identifier}$, *EnterTime, LeaveTime*).

The data gathering application used for the BLIP data set is not designed to handle devices that are tracked in more than one location at a time. This means that when a device is located in an area that is covered by more than one access point, the data gathering application creates bounce records.

A real-time data warehouse is a system which reflects all changes in its source, or sources, in real-time or near real-time. Since the data available in a traditional data warehouse is not always up to date, a DW is typically used for analysis and long term improvements to overall business strategies and tactics. A RTDW is different, as changes are reflected to the systems in (near)real-time. The operational systems are designed to accept inputs or changes to data regularly, hence have a good chance of being regularly updated. Knowledge workers can, within the scope of right-time, analyze and take action if something needs to be addressed.

The need for data being processed at a quicker pace in a RTDW, than a traditional DW, enforces changes to the way data from the OLTPs are being processed. The traditional bulk loading at fixed nightly or weekly intervals does not support the need for real-time or near real-time data. Instead, an approach where data is continuously being fed to a software system serves the same purpose of an ETL.

The first sections have given us knowledge and understanding of the different concepts and guide lines for designing and implementing real-time data warehouses. We put that knowledge to use by designing a real-time data warehouse that suits the business needs of the BLIP case. Two data warehouse designs are presented, a specific and a generic, and the generic data warehouse design is chosen to be the implemented warehouse, as it provides support for more complex queries. While the support for more complex queries comes at the expense of a slower data warehouse, we consider flexibility and expandability more important than the ability to make very specific queries fast.

We implement a CTF to extract data from the OLTP, transform it and lastly, save the transformed data in a MSSQL Server. To capture the changes made in the BLIP OLTP, an insert trigger is created on the main table which stores the new records into another table. The CDC module deletes records from this table once they have been pushed into the CDC queue. The task of the transformer is to check all records, which are in the correct format as verified by the data cleansing module in the CDC, for valid moves and bouncing records. The transformer is in charge of checking the validity of the moves within the airport, and perform bounce detection and elimination. To ensure scalability of the system, the transformer is coded in a way that supports multi-threaded execution.

With a $Bounce_{threshold}$ set to 7 seconds, the amount of tracking records is reduced from 25,000 to 9,082, which is equal to a reduction of tracking records by 63.67%. In comparison, the bounce detection and elimination algorithm presented in the pre-master thesis removed 2,642,095 out of 21,161,406 using the same threshold, which equals a reduction of 12.49%.

To test the response time of the real-time data warehouse and the impact the choice of the storage model have on the system, we have implemented the cube with two different storage models in the Microsoft SQL Server Analysis Services 2008. The first is a pure ROLAP cube, where all detail data, dimensions, and aggregations are stored in a relational format, where the second is implemented using the real-time HOLAP proactive caching setting and the HOLAP storage model, i.e. detail data is stored in a relational format, while the aggregations are stored in a multidimensional format. Even though the HOLAP cube provides slightly better response times, we still estimate that the pure ROLAP implementation is preferable.

We have shown how the implemented solution can be used to answer the question, *Are there any congested areas in the airport?*, and the question 2, *Are there any congestions forming?*, by executing MDX queries and using the results to visualize the state of the environment. To optimize the execution time of the queries, the data warehouse can be customized to support the aggregations needed to calculate the results of specific queries faster.