A Study of Open Payment Fare Systems: System Design, Fare Engine Algorithm and GTFS Extension

by

Yin Wang

M.S. Control Science and Engineering
Tsinghua University, 2011

Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of Master of Science in Transportation at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

© 2014 Massachusetts Institute of Technology. All rights reserved.

Signature redacted

Author ........................................

Department of Civil and Environmental Engineering
May 12, 2014

Signature redacted

Certified by .................................

George Kocur
Senior Lecturer of Civil and Environmental Engineering
Thesis Supervisor

Signature redacted

Accepted by .................................

Heidi M. Nphf
Chair, Departmental Committee for Graduate Students
A Study of Open Payment Fare Systems: System Design, Fare Engine Algorithm and GTFS Extension

by

Yin Wang

Submitted to the Department of Civil and Environmental Engineering on May 12, 2014, in partial fulfillment of the requirements for the degree of Master of Science in Transportation

Abstract

This thesis describes the design and implementation of the key parts of an open payment system that supports mobile phone ticketing for the Long Island Railroad (LIRR), a part of the New York Metropolitan Transportation Authority (MTA). While many public transit agencies across the world are still using traditional fare systems, open payment system can help reduce lifecycle costs for transit agencies while making public transit service more convenient to passengers. One of the keys to the implementation of an open payment fare system is to infer trips and compute fares from a series of taps on gates and fareboxes by an open payment device, either a bankcard or a mobile phone.

A trip construction algorithm based on a finite state machine is proposed to automatically group tap events from a single user into trip segments according to the MTA’s fare rules and send them to a fare engine for fare calculation. The trip construction algorithm (implemented in the trip server) can handle bus, subway and railroad tap events in the MTA’s system with fraud detection and exception handling.

The fare engine adapts a label-correcting shortest path algorithm to find the chosen paths for each trip segment and to calculate the fare based on the LIRR’s fare structure, including a number of configuration parameters such as minimum fare, minimum transfers and minimum travel time. The shortest path algorithm runs on a directed graph that is capable of modeling LIRR’s complex service and transfer restrictions.

Recognizing the limitations of system-specific fare engine design, this thesis also proposes extensions to the General Transit Feed Specification (GTFS), and develops a generic fare engine design that can be shared across multiple transit systems. These extended designs are studied and tested on the LIRR and Transport for London (TfL) networks. The proposed design appears to accommodate the fare policies of many transit systems; eight systems are briefly reviewed.
Thesis Supervisor: George Kocur
Title: Senior Lecturer of Civil and Environmental Engineering
Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. George Kocur, for his excellent guidance, patience, and always providing me with gems of expertise and timely insight. His continuous support helped me at all times during the research and writing of this thesis.

This research is funded by Nokia’s Transit Platform and Content Department as part of its collaboration with MIT. I would like to acknowledge Justus Brown and Peter Preuss from Nokia for the support they have lent to this project.

I want thank all of my colleagues in the MST program for their support. I am also grateful to Kris Kipp, who’s always there to help me with administrative issues.

Finally I owe the deepest appreciation to my parents. They are always supporting me and encouraging me with their best wishes.
Chapter 1

Introduction and Background

1.1 Open Payment System

Definition: Near Field Communication (NFC) is a set of short-range wireless communication technology standards for smartphones and similar devices.

Definition: A Tap is a transaction with a system time stamp and other necessary information that is sent from the card/phone to the reader and then transferred to the server from the reader, or it may be sent directly from the phone to the server. Such a transaction will contain information on the location and time of the tap event, user identification number, reader ID and other information as required.

Definition: A reader is a device that can start the communication with an NFC tag. The reader processes taps and sends them to a server; it can be incorporated into different devices such as a point-of-sale device or a device mounted on a gate or farebox where a rider taps their phone or bankcard.

Many transit agencies across the world are still using proprietary fare payment systems implemented years or decades ago, which impose significant administrative responsibilities on the transit agencies, as well as often being difficult for infrequent travelers or visitors to understand and use. These systems may be expensive to maintain, making it hard for transit agencies to reduce costs, upgrade system components and generate new revenue streams (Smart Card Alliance, 2006) (Lau, 2009).

Open payment systems use open standards and open architectures, which allow
transit agencies to select generic components from a large number of vendors, and replace/upgrade them at different times independently. This reduces the lifecycle cost for transit agencies and makes it more convenient for travelers moving around the world, who no longer must worry about reading the tariff, obtaining fare media and choosing the right ticket type.

Mobile payment has been implemented mostly to sell entertainment products, such as digital content or codes to access gaming sites. However, business-oriented cases have been launched quite successfully, including ticketing for bus and subway systems (National Retail Federation, 2011).

A typical design for a public transit open payment fare system is shown in Figure 1-1 (Kocur, 2010).

In this system, the rider carries a contactless credit or debit card or Near Field Communications (NFC) enabled phone, which he/she can tap at a point of service, usually a card reader at a subway gate or bus fare box. The reader reads the information on the card, sends it to a server to record the time and place of the tap event. Unlike the use of a credit or debit card in retail stores, a single tap in a public transit network alone does not usually qualify as a transaction. A group of taps at origin.
station, destination station and sometimes intermediate points together form a complete trip with a price. The server runs a trip/fare engine to group the taps into trips and calculate the fare for each trip according to the system-specific tariff and send it to the billing engine. The billing engine sends the bill to the merchant acquirer (the same entity that processes credit and debit card payments for merchants), who gets authorization from the payment network (such as Visa or MasterCard), and finally the bill is posted to the card the rider carries.

All cities have unique public transport networks and fare structures, most of which are not readily understandable to infrequent travelers or visitors. With NFC phones and readers placed at public transport stations (rail, subway, bus), passengers no longer need to obtain fare media or choose a ticket type. With NFC and open payment (discussed below), the traveler taps the phone at the gate or farebox, rides the public transport service and taps out when he/she leaves the system. The phone records the origin station and destination station (and intermediate points if required by the fare policy), calculates the best fare for this trip, and charges that amount to the payment account associated with the phone.

The technology simplifies the ticketing procedure for both passengers and transit agencies. It saves the passengers from having to learn the tariff to figure out which fare they should buy when they travel in a new city. It moves most fare payment and inspection off the train for the LIRR, and makes it a self-service function for the customers, with only random inspection required.

Recognizing the value of using open payment system in the mass transit marketplace, transit agencies throughout the world have implemented or plan to implement open payment in their service systems, seeking to improve customer service and operating efficiencies (Smart Card Alliance, 2011b).
1.2 Mobile phone payment systems, open and non-open payment

Definition: mTickets are virtual tickets issued by ticket issuer to its customers, which serve the same purpose as a traditional paper ticket. The mTickets are held on user’s mobile phone after purchase, usually delivered as SMS messages, pictures, barcodes, etc.

Masabi is one of the leading companies in transit mobile ticketing and payments. Masabi’s JustRide system allows the passenger to buy and display tickets on their mobile phones, either as 2D barcodes or mTickets, and has been deployed in several transit systems, including Boston, San Diego, and London (Masabi, 2013).

Boston’s Massachusetts Bay Transportation Authority (MBTA) is the first transit agency in the United States to adopt mobile ticketing. The MBTA began offering mobile tickets to its commuter rail customers in 2012, using the JustRide mobile ticketing platform from Masabi. This app allows customers to purchase and display a mobile ticket to the conductor on the train as well as check schedules, maps and service alerts. The tickets are displayed on the phone’s screen as a digital “flash pass” or encrypted barcode (MBTA, 2014).

New Jersey Transit has launched an app called MyTix in April 2013 that enables mobile ticketing. MyTix is now available for all NJ Transit rail lines. The customer can purchase tickets and passes on his or her phone; he or she must activate the ticket prior to boarding the train and display the activated ticket to the conductor (NJ Transit, 2013). Other similar systems include the GoPass Mobile Ticketing Application for Dallas Area Rapid Transit (DART) provided by Unwire (DART, 2014) and TriMet Mobile Ticketing at the Tri-County Metropolitan Transportation District of Oregon (TriMet) provided by GlobeSherpa (TriMet, 2014).

Both the MBTA and NJ Transit mobile ticket app require a conductor to examine the validity of the tickets, while in London, the ticket gates are equipped with barcode readers that can identify a valid barcode ticket on mobile phones. The first barcode ticket gate for mobile tickets with Masabi was launched in the UK in 2010 for Chiltern
Railways (Total Rail, 2014). Customers can purchase tickets in the mobile app until 10 minutes prior to departure and use the phone as the ticket. After activating the ticket on the phone, a 2D barcode will be displayed on the phone’s screen and the customer can scan the ticket at the gate to enter the station (Chiltern Railways, 2014).

Several transit agencies have deployed mobile ticketing for their bus services.

The Austin based Capital Metro launched a mobile ticketing app for its local buses and MetroRapid service in 2014, created by Bytemark. The app allows customers to purchase tickets using the app and show the smartphone screen to bus operators or scan the on-screen barcode at the farebox upon boarding the bus (Capital Metro, 2013).

A survey and demand analysis conducted in Brakewood et al. (2014) suggests that mobile ticketing appears to be a compelling alternative to traditional ticketing methods, and its adoption by rail operators and utilization by riders is likely to increase in the near future.

Most of the mobile ticketing schemes in operation today rely upon technologies such as 2D barcodes or readable mTickets. However, the GSM Association (GSMA), which unites nearly 800 of the world’s mobile operators with 250 companies in the broader mobile ecosystem to support the standardising, deployment and promotion of the GSM mobile telephone system, clearly identified NFC as the technology of choice for smart ticketing in GSM Association (2011).

The existing approaches in mobile ticketing all require the customer to pre-purchase a specific type of ticket before riding the service, which requires the customer to have good knowledge of the system’s service and fare structures, and thus is not convenient for tourists or infrequent users of transit systems. The NFC ticketing approach described in this thesis doesn’t require any action from the customer before or after riding the service: they simply tap in and tap out of the system and the fare is automatically calculated and charged to their account. Also, most non-NFC approaches require manual inspection of tickets on board and do not work in gated environments. This limits interoperability across gated and non-gated modes, and it
may require additional staff on trains solely for ticket inspection.

NFC mobile contactless payments take advantage of the installed infrastructure for contactless bank card payments, such as the readers deployed in TfL’s system. MasterCard provides MasterCard PayPass to its users that allows them to tap the mobile phone on a point-of-sale terminal reader to pay at PayPass merchants (MasterCard, 2014). Google announced Google Wallet in October 2011 and the service is now available across the United States. Customers can use Google Wallet on their NFC enabled mobile phones to shop at merchants who accept contactless payment (Smart Card Alliance, 2011b).

Using the same process that supports mobile contactless retail payment, NFC-enabled mobile phones can also be used for transit ticketing. Juniper research provided a five year forecast for mobile ticketing in a whitepaper in 2012, stating that NFC is a major new technology advance in the mobile commerce market which is poised to radically change the mobile ticketing market and that it will advance over the forecast period to service half the market by 2016 (Juniper Research, 2012).

Standards and hardware availability are paving the way for NFC mobile payments (Smart Card Alliance, 2011a). NFC already has widespread support from device makers. According to a report by Berg Insight, global sales of NFC handsets rose 300% to 140 million units in 2012. The research firm also expects one billion NFC handsets to be shipped in 2017. At the same time, the number of NFC point-of-sales terminals is also set to rise dramatically (GSM Association, 2014).

Both MasterCard and Visa announced support for a way to enable card-like applications via NFC, Host Card Emulation (HCE), which is supported in Google’s latest version of Android, 4.4 KitKat (Rey, 2014). HCE emulates the security element of NFC smart card in hardware as software without relying on the Trusted Service Manager (TSM) and Mobile Network Operators (MNO), thus allowing any developer to include payment in their app from the cloud. This will allow NFC payments to bypass the carrier restrictions and is believed to boost the development of NFC mobile payment. In Paris, the transit authority STIF deployed a new generation of Passe Navigo, the travel pass used by six million commuters in the greater Paris region with
an NFC chip in January 2014. This move is targeted at replacing passengers' paper tickets in the first implementation and will pave the way for NFC phones to be used as travel tickets on the Paris area's buses and trains as well as the Paris Metro in the future (Kleiber, 2014).

Transport for London has installed contactless open payment readers upgradeable to NFC in their 8,500 buses since 2012. The adoption of open payment technology allows customers with a contactless bank card to tap the card at the reader and pay the fare. Open payments account for 20% of all bus payments in TfL's system (Boyle, 2013).

Through the various field trials and full system implementations of mobile ticketing across the world, it has been continually demonstrated that transit patrons warmly embrace mobile fare payment and associated real-time information services (Smart Card Alliance, 2012).

1.3 Problem Statement and Research Question

Most of the parts in the open payment fare system can be shared across multiple transit agencies or outsourced to vendors who serve non-transit merchants. But the trip server and fare engine need to be system-specific to be able to map a set of taps into a group of trip segments, and trip segments into fares, because the network structure and tariff of each transit system varies in many ways. Thus, this research focuses on the design and implementation of the trip segmentation and fare calculation function in an open payment fare system for transit services.

Nokia conducted a NFC Mobile Phone Pilot for fare collection, in partnership with LIRR from 2011. The system allowed passengers to pay fares by tapping their NFC mobile phones against a "touch point" — a small disc-shaped device at the station where they began their trip, and tapping it again when they arrived at their destination station. (LIRR stations are not gated, and current practice uses paper tickets and 100% on-board inspection.) The passenger's account is billed for the fare based on the LIRR fare structure. The first stage of the project tested the system
design of collecting taps from mobile phones and computing the appropriate fare for each trip. Actually billing the fares to a payment account was not included in the test.

This thesis makes use of the tap data collected from the project and mainly focuses on the design and implementation of a trip server and a fare engine for the LIRR mobile ticketing pilot. In this research, we develop solutions for challenges arising from both the complexity of LIRR’s fare structure and the requirement of a generalized fare engine in future open payment fare systems.

The thesis is structured as follows:

Chapter 2 describes the system design in detail: the system requirements and a complete design of system structure are proposed in section 2.1 and 2.2; the data structure and basic functionality of the tap server is discussed in section 2.3; section 2.4 describes the detailed design of the trip server, focusing on trip segmentation using a finite state machine; the modeling of system network as a directed graph and the development of the shortest path algorithm is presented in section 2.5.

Chapter 3 evaluates system performance with various test cases and examples.

Chapter 4 discusses the possibility of a general fare specification format and the development of a general fare engine that will serve multiple transit service systems. An extension to GTFS is proposed and a general fare engine design is tested on LIRR’s and TfL’s networks.
Chapter 2

System Design

2.1 System Requirements

The main function of the system is to:

(1) Collect taps from passengers’ mobile phones;  
(2) Validate the taps;  
(3) Group taps of the same passenger into trip segments.  
(4) Calculate the appropriate fare for each of the trip segments.  
(5) Return the fare information to the mobile phone application.  

This research focuses on functions (3)-(5) while including some necessary exception handling in each stage.

2.2 System Structure Design

The proposed system structure is divided into four components, as illustrated in Figure 2-1.

The system consists 4 main components:
• the mobile phone (phone application) that the user holds and taps at gates and validators;
• the tap server that keeps a record of all tap events it receives;
• the trip server that groups tap events of a single user into trip segments; and finally
• the fare engine that calculates the fare for each trip segment.

The separation of system functions provides the flexibility in Public-Private Partnership (PPP): certain transit agencies may want to keep the trip segmentation and fare calculation under their own control while some others may decide to outsource the entire service to a private sector.

A brief description of the system’s functions, in time sequence, is:

1. [Mobile Phone]
   Client application on the phone collects taps at touch points with station ID, time of tap and any other required information. This information is stored on the phone to be transmitted to the tap server.
2. [Mobile Phone]→[Tap Server]
   The client application sends all the taps it has, which don't have either evidence or price or trip id, as described below, to the tap server in the form of a HTTP request.
   This step corresponds to the "POST(tapo)" event between mobile phone and tap server in Figure 2-1.

3. [Tap Server]
   The tap server listens for newly arrived taps. After receiving new taps, the tap server will calculate "evidence" for each new tap. "Evidence" is a field associated with each tap. Valid evidence indicates that this tap is recognized as a valid tap with no fraud issues. Taps missing certain information and the ones regarded as being invalid will not have the "evidence" field filled out by the tap server, and will not be sent to the trip server. Evidence is always calculated whether this tap is sent to the trip server or not. The tap server has a copy of all the taps it has received, regardless of their validity, in the database.
   This step corresponds to the "calculate evidence" event at the tap server.

4. [Tap Server]→[Trip Server]
   The tap server sends all the taps (of a single client/account) without price or trip ID to the trip server for trip segmentation, sorted by the time of tap. This step corresponds to the "POST(tapo)" action between the tap server and the trip server.

5. [Trip Server]
   The tap sequence for each customer received from the tap server is segmented on the trip server into separate trip segments. Grouping taps into trip segments is a partial operation on the trip server, e.g., there could be a tap_in without a tap_out because the trip may still be in progress. Such segmentation is done through a state machine customized for the fare policy of the individual public transport system. The trip server does not mirror the data on the tap server.
   This step corresponds to the "group into trip segments" event at the trip server.
6. [Trip Server]→[Fare Engine]
   The trip server sends each trip segment it gets to the fare engine to calculate
   the individual fare. This step corresponds to the “POST(trip[])” event between
   the trip server and the fare engine.

7. [Fare Engine]
   The fare engine calculates the fare for each trip segment it gets based on the
   specific network and fare structure. This information about the transit system
   is stored on the database connected to the fare engine server.
   This corresponds to the “calculate fare” event at the fare engine.

8. [Fare Engine]→[Trip Server]
   The fare engine will return the best price, or an invalid trip segment warning
   message, for each trip segment to the trip server.
   This step corresponds to the “trip[](price)” event between the fare engine and
   the trip server.

9. [Trip Server]→[Tap Server]
   The trip server will return the trip ID and price for each tap to the tap server.
   This step corresponds to the “tap[](tripID, price)” event between the trip server
   and the tap server.

10. [Tap Server]
    The tap server updates the price and trip ID fields for each tap in the database.

11. [Tap Server]→[Mobile Phone]
    The tap server will return taps with valid trip ID, price and evidence to the
    client. Invalid taps are not returned.
    This step corresponds to the “tap[](evidence, tripID, price)” event between the
    tap server and the mobile phone.

All eleven steps appear to execute within a single POST from the passenger’s
point of view. There are two options to send the collected taps to the tap server: (1)
sending taps directly from the phone to the tap server; (2) and sending taps from
gates to the tap server.

In the first option, the phone is required to have an internet connection to be able
to send the tap sequence to the tap server. The mobile phone application will always
send all the taps without evidence, trip ID or price on the phone to the tap server.
Using this design eliminates the case where, due to network or phone problems, an
d earlier tap arrives at the tap server after later taps have already been processed by
the trip server. This design guarantees that the trip server receives all taps in the
correct time order and does not need to worry about having previous taps suddenly
joining the tap sequence.

The out-of-sequence tap problem exists in the second option when gate readers
rather than phones are used to send taps to the tap server. Imagine a scenario that the
user taps the phone at Gate A when he/she enters the station, boards the train, and
then taps at Gate B to get out of the station when he/she reaches his/her destination.
Gate A will send the first tap to the tap server while Gate B will send the second
tap to the tap server, but there's no guarantee that the first tap will arrive at the tap
server before the second tap. Possible causes of such problems may be different tap
volume handled by each gate, different internet bandwidth at each gate, etc.

Although the second option doesn’t require internet access on the phone, taps
arriving at the tap server in the wrong order may cause problems to subsequent trip
segmentation and fare calculation. Thus the first option, sending all taps without
evidence, trip ID or price directly from the phone to the tap server at once, is adopted
in this research. The first option also allows the use of passive tags, without power
or communication, as is appropriate for nongated stations on the LIRR.

2.3 Tap Server

As shown in the system’s design in Figure 2-1, the mobile phone application can
only communicate with the tap server so the multiple steps appear to execute within
a single POST from the passenger’s point of view. The implementation of the tap
server is not the focus of this research work and only a brief description of the tap server is given here.

The tap server is connected to a database, which has a full copy of all the taps that have been received from all the passengers in a single table. The design of the tap table is shown in Table 2.1.

The mobile phone application will send all the taps of a single user which don’t have either evidence or price or trip ID to the tap server in a single string formatted as a JSONArray in a HTTP POST request. The format of the request string is shown as follows. The sample JSONArray consists of 2 taps; the data is an example only. Some fields are eliminated for clarity.

```json
{
  "taps": [
    {
      "t.time": "2012-06-25 09:07:16",
      "u.id": "352425050850851",
      "t.type": "1",
      "chall": "97572c082f4a2244fc6afa9849db36695d94c44",
      "sign": "7d059b72018c6ad01ba1c5c131298f637c758aa2",
      "card.r": "3834306472386d6",
      "cert": "043c8c3aa62780",
      "ctr": "51",
      "phone": "2b343931353135353135343032",
      "event.type": "TAP_IN",
      "sso_uname": "4c5334313140616",
      "sso_uid": "6330313532623664",
      "sso_token": "6e71517a423658613579683",
      "tripID": "31"
    },
    {
      ...
    }
  ]
}
```
"t_time":"2012-06-25 09:48:22",
"u_id":"352425050850851",
"t_type":"1",
"chall":"Off086b1a6f753ae3edf0fe77ef236deed37e00",
"sign":"f8bd5562c918b8d8f2b8e290259487dabaa7d252",
"card_r":"38343064723572752d3",
"cert":"04578a3aa62780",
"ctr":"52",
"phone":"2b34393135313535313535343032",
"event_type":"TAP.OUT",
"sso_uname":"4c5334313140616f6c2e",
"sso_uid":"63303135326236642d",
"sso_token":"6e71517a42365861357",
"tripID":"32"
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Integer</td>
<td>Primary key, unique id for each tap</td>
</tr>
<tr>
<td>imei</td>
<td>String</td>
<td>Phone IMEI number</td>
</tr>
<tr>
<td>t_time</td>
<td>String</td>
<td>Time of tap, in the format of “YYYY-MM-DD HH:MM:SS”</td>
</tr>
<tr>
<td>rec_time</td>
<td>String</td>
<td>Time when the tap is received by the tap server, in the format of “YYYY-MM-DD HH:MM:SS”</td>
</tr>
<tr>
<td>u_id</td>
<td>String</td>
<td>Phone ID</td>
</tr>
<tr>
<td>t_type</td>
<td>Integer</td>
<td>“1” for gated system, “0” for non-gated system</td>
</tr>
<tr>
<td>chall</td>
<td>String</td>
<td>Non-Gated System: Challenge chall is generated by the phone and sent to the touch point; Gated System: Challenge chall is generated by the touch point and sent to the phone.</td>
</tr>
<tr>
<td>sign</td>
<td>String</td>
<td>ObC signature. See Nokia documentation on ObC.</td>
</tr>
<tr>
<td>card_r</td>
<td>String</td>
<td>Station ID, different from those used by fare engine.</td>
</tr>
<tr>
<td>cert</td>
<td>String</td>
<td>Ticketing key certificate of the phone</td>
</tr>
<tr>
<td>ctr</td>
<td>Integer</td>
<td>Non-Gated System Only: Counter value which is incremented at every successful tap event</td>
</tr>
<tr>
<td>phone</td>
<td>String</td>
<td>Phone number of the user received from Nokia SSO. Nokia SSO is a single sign-on account which allows user to access all Nokia's service.</td>
</tr>
<tr>
<td>event_type</td>
<td>String</td>
<td>Event type suggested by the client, to be overwritten by trip server: “TAP_IN”, “TAP_OUT”, “SINGLE_TAP”, “PENALIZED_TAP”</td>
</tr>
<tr>
<td>sso_uname</td>
<td>String</td>
<td>Nokia SSO user name which is login email address</td>
</tr>
<tr>
<td>sso_uid</td>
<td>String</td>
<td>Nokia SSO user id</td>
</tr>
<tr>
<td>sso_token</td>
<td>String</td>
<td>Nokia SSO token to verify the user</td>
</tr>
<tr>
<td>checked</td>
<td>Integer</td>
<td>Value that represents the verification of the tap record. Usually 1 for success</td>
</tr>
<tr>
<td>discount_group</td>
<td>String</td>
<td>Discount group of the client: “ADULT”, “CHILD”, “SENIOR”.</td>
</tr>
<tr>
<td>price</td>
<td>Integer</td>
<td>Price for the trip that this tap belongs to.</td>
</tr>
<tr>
<td>trip_id</td>
<td>String</td>
<td>Trip ID assigned to the trip that this tap belongs to.</td>
</tr>
</tbody>
</table>

Table 2.1: Database table design of taps
2.4 Trip Server

The main function of the trip server is to group taps of the same user into separate trip segments according to system specific segmentation rules and then pass the trip segments to the fare engine for fare calculation.

The trip server does not have access to the tap server’s data. It listens to a specified port for HTTP POST requests from the tap server, and groups the tap sequence of a single user into tap pairs belonging to separate trip segments. This procedure is called “trip segmentation” and is done using a finite state machine with pre-set transition rules. This state machine is designed specifically for a city’s public transport system because there may be different rules about transfers, maximum trip durations and other issues in different cities.

The finite state machine for trip segmentation is triggered by tap events. For the LIRR system, there may be three types of taps in the system:

1. **Station Taps (ST)**: taps at station gates when entering or exiting the LIRR system. All valid rail trip segments should start and finish with station taps.

2. **Intermediate Validator (IV)**: readers placed on the platforms of some of the stations. Passengers can tap their phones at these intermediate validators to provide extra information on their trips to the transit agency, such as transferring at certain stations.

3. **On-board Inspection (OB)**: on-board inspection devices are provided to conductors to inspect whether passengers have valid tickets/taps.

For the purpose of the pilot project, only station taps are collected from the volunteers and included in the state machine. Intermediate validator and on-board inspection taps can be added to the state machine when real data becomes available.

For completeness, when designing the state machine for the LIRR system, the MTA subways and buses are also taken into consideration. Bus taps (BUS) and subway taps (SUB) are also included in the design of the state machine.
Passengers are required to tap into and tap out of the LIRR system when they ride a train. Trips in the LIRR system are generated by grouping two consecutive station taps together. MTA bus and subway services only require a single tap before riding the service so each bus or subway tap is considered as an individual trip, since riders only tap in and do not tap out. The state transition diagram of the state machine is shown in Figure 2-2. Transfers along the LIRR are not modeled explicitly; the rider only taps in at the origin and taps out at the destination, and there are no taps at transfer stations. Trips with transfers will be handled by the fare engine when calculating the fare.

2.4.1 Tap Events

The tap events handled on the trip server are:

- "ST": Station taps at station gates in LIRR system;
- "BUS": Taps when boarding a MTA bus.
• “SUB”: Taps at the gates when entering a MTA subway station.

2.4.2 States and the Transition Logic for Trip Segmentation

• Initial State

• Valid Rail Trip Started
Starting from the initial state, an “ST” is identified as a tap-in and a valid rail trip has started. This “ST” is regarded as the origin of the trip. A trip ID is automatically generated for this trip. If an “ST” tap arrives in this state with the same station id and the time gap between this new tap and the previous “ST” tap is smaller than a pre-set threshold, then this new tap is considered as a duplicate, which will not trigger the transition to another state. There are cases where a passenger has multiple taps at the same station with small time gaps (20 seconds or so). These duplicate taps may be the result of tapping at the same reader several times, e.g. in the case of reader failure, or the passenger being unsure if he or she tapped successfully.

• Valid Rail Trip Completed
If the previous state is “Valid Rail Trip Started”, and (1) the new event is “ST”, (2) the new tap is from a different station than the previous tap (or the new tap is at the same station as the previous tap but the time gap is bigger than the pre-set threshold), this means the passenger has tapped out of the system and the trip is completed. This new “ST” tap is regarded as the tap-out and the destination of this rail trip. Tap pairs for this rail trip will be sent to the fare engine to calculate the fare. There are three possible responses from the fare engine server:

(1) Trip Price Generated Successfully. The tap pair for this trip is valid (there is a route in the system between the origin and destination stations and the time gap is within the longest allowed travel time) and a price is returned to the trip server. The state machine will automatically transfer to the
end state afterwards.

2) Invalid Tap. One of the taps in the trip segment does not belong to the system (e.g., taps at NFC advertisement board or taps between NFC phones, etc.). The fare engine will return a string message specifying which tap it is and the state machine will mark this tap as an “INVALID_TAP” with price $0 and re-do the trip segmentation with the rest of the taps (including the ones that belong to the current trip).

3) Failed to Generate Trip Price. There’s no route available between the origin and destination station in the system, or the time gap between taps has exceeded the longest allowed travel time. This means the taps are mismatched or there may be some fraud issues. The fare engine will return an error message to the trip server. The trip server will mark the first tap in this trip as a penalized tap and the second tap as unprocessed, send the penalized tap to the fare engine for pricing, and then automatically transfer to the end state.

- Valid Bus or Subway Trip Completed
A “BUS” or “SUB” tap is received after the initial state and a valid bus or subway trip is started/ended (passengers only need to tap into the bus or subway system and do not need to tap out, so the bus or subway tap is considered as a tap-in and a tap-out at the same time). This state can also be triggered when a BUS/SUB tap arrives in the state “Valid Rail Trip Started”. In this case, first the previous “ST” tap is marked as a penalized tap and passed to the fare engine for pricing, and then the BUS/SUB tap is passed to fare engine for pricing.

1) In the current system, all subway or bus trips are charged a single fare, so there is no exception handling for bus/subway trips in the fare engine.

2) Free bus/subway transfers are not included in the current implementation. There are free transfers between bus and subway trips in MTA system, but the rules are complicated. Since we do not have actual bus/subway taps
in our pilot data, the free transfer logic is left out of the state machine design. Therefore, passengers will be charged for every bus/subway trip they make.

2.4.3 Example

A passenger travels in the LIRR system from station A to station B, and later in the day from station C to station D. The tap server should receive four consecutive "ST" taps, but the passenger forgets to tap out at station B. Thus, what the tap server actually receives is a tap sequence of "A, C, D" instead of "A, B, C, D". In this case the state machine will first group "A-C" as a valid rail trip segment and pass it to the fare engine for pricing.

If the there's no route available from A to C, the fare engine will return an error message to the trip server. The state machine will mark tap "A" as a penalized tap and pass it to the fare engine for pricing. Tap C will be marked as unprocessed; the state machine will go back to initial state and continue with tap C. This time it will group tap C and D together correctly.

\[ A, C, D \rightarrow A \text{ (penalized), C-D} \]

It is also possible that there is a route between station A and C in the system and the fare engine manages to generate a fare for this tap pair "A-C". In this case the trip server can only accept "A-C" as a valid trip and continue with tap D because there's no way to know whether there's a missing tap between tap A and C.

\[ A, C, D \rightarrow A-C \]

A future improvement is to compare the time gap between paired taps and the system's default travel time when such information is available. If the difference is above a threshold, the fare engine may reject this tap pair and return an error message. (If major delays in train service are occurring, this edit might be turned off.) If tap A and tap C are matched incorrectly, this may generate errors in all future taps, which should be a rare case. Since most passengers travel only during a short portion of the day, the mistake can be corrected when the passenger leaves the system, and the time gap between two consecutive taps exceeds the maximum travel
This mismatch can also be corrected if at some time point a single “ST” is followed by a “BUS” or “SUB”. The following are some examples of how the state machine will act to sequence taps. Taps in () stand for missing taps. Underlined taps represent penalized taps.

- A-B, C-D, E-F, BUS → A-B, C-D, E-F, BUS
- A-(B), C-D, E-F, BUS → A-C, D-E, F, BUS
- A-(B), C-D, E-F, BUS → A, C-D, E-F, BUS (if there’s no route between A and C)
- A-(B), C-D, E-F, BUS → A-C, D, E-F, BUS (if there’s route between A and C but the time gap between D and E exceeds the maximum travel time allowed in system)

2.4.4 Request and Response Data

The request from the tap server to the trip server is also in a JSONArray format shown below. The JSONArray consists of 2 taps; the data is an example only:

```json
{“tapsShorted”:{

{“t_type”:“1”,
“card_r”:“3834306472386d6”
“event_type”:“TAP_IN”,
“gtfs_type”:“2”,
“discount_group”:“ADULT”

```
The response from the trip server to the tap server is also formatted as a JSONArray shown as follows. The JSONArray can be empty if the tap sequence belongs to an unfinished trip segment. Trip ID is generated by attaching tap ids belonging to the same trip one after another with "#" mark.

```json
{
  "tapsGrouped":null
}

{
  "tapsGrouped":[
    {
      "id": "789",
      "event_type": "TAP_IN",
      "trip_id": "789#790",
      "price": "230"
    },
    {
      "id": "790",
      "event_type": "TAP_OUT",
      "trip_id": "789#790"
    }
  ]
}
```
2.5 Fare Engine Server

The main function of the fare engine is to derive the path for a given trip segment and decide the appropriate fare for this trip. The fare engine is separated from the tap and trip servers because transit agencies may want to keep the fare calculation under their own control.

2.5.1 Database Schema

The fare engine is connected to a database which has a full description of the transit system's network and fare structure. The fare engine will use these data to build up the transit network and get the path and fare for any given trip segment.

Every station belongs to one and only one fare zone in the LIRR system. There may be more than one type of service run between the same station pair, which is defined in the "ServiceLinks" table.

Fare types are defined by the inner most zone and outer most zone traveled on the trip, along with the zone where the transfer is made for indirect trips. A single fare type may cover trips with different inner-outer zone combinations, which is defined in the "FareStructure" table. Fare also varies for peak and off-peak trips with the same zone range traveled, depending on the time the train leaves/arrives at New York City. The critical times for deciding whether a train is peak are stored in the four peak time tables.

Detailed information on each table in the database is described as follows.

1. Stations
Figure 2-3: Database Entity Relationship Diagram.
Table 2.2: Table of stations in the LIRR system.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StationID</td>
<td>int</td>
<td>Primary key, start from 0 for convenience. This is actually a reader ID, and there may be more than one reader per station. Reader IDs are mapped to a station number.</td>
</tr>
<tr>
<td>StationName</td>
<td>varchar</td>
<td>Station name</td>
</tr>
<tr>
<td>Zone</td>
<td>int</td>
<td>Zone this station is in, references ZoneID in table “Zones”</td>
</tr>
<tr>
<td>LIRRStationID</td>
<td>int</td>
<td>Station ID in the LIRR system. Different than the station number to which reader IDs are mapped.</td>
</tr>
<tr>
<td>EntryEnabled</td>
<td>int</td>
<td>Whether entry to the system is open at this station, 1 for entry enabled.</td>
</tr>
<tr>
<td>ExitEnabled</td>
<td>int</td>
<td>Whether exit from the system is open at this station, 1 for exit enabled.</td>
</tr>
<tr>
<td>TransferEnabled</td>
<td>int</td>
<td>Whether transfer is allowed at this station, 1 for transfer allowed.</td>
</tr>
</tbody>
</table>

2. Zones

Table 2.3: Table of fare zones in the LIRR system.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZoneID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>ZoneName</td>
<td>varchar</td>
<td>Station Name/ID in LIRR system</td>
</tr>
</tbody>
</table>

3. Services
Table 2.4: Table of service branches in the LIRR system.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceID</td>
<td>int</td>
<td>Primary Key</td>
</tr>
<tr>
<td>ServiceName</td>
<td>varchar</td>
<td>Service name for line or branch</td>
</tr>
</tbody>
</table>

The LIRR system is made up of eleven passenger branches or lines (as shown in Figure 2-4):

- City Terminal
- Babylon Branch
- Ronkonkoma Branch
- Port Washington Branch
- Far Rockway Branch
- West Hempstead Branch
- Hempstead Branch
- Oyster Bay Branch
- Long Beach Branch
- Port Jefferson Branch
- Montauk Branch

For scheduling and advertising purposes, some of the branches are further divided into sections such as the case with the Montauk Branch and the Babylon Branch. Both branches are referred to as Babylon Branch in our system for simplicity. There is service running in both directions on each branch: westbound (toward New York City) and eastbound (from New York City).
4. FareTypes

Table 2.5: Table of fare types provided in LIRR system.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FareTypeID</td>
<td>int</td>
<td>Primary Key, start from 0</td>
</tr>
<tr>
<td>FareTypeName</td>
<td>varchar</td>
<td>Fare Name in LIRR system (e.g., Adult Peak)</td>
</tr>
</tbody>
</table>

The main fare types that are covered by the system now include:

- one way peak,
- one way off-peak,
- AM peak or off-peak senior/citizen disabled,
- peak child, and
- off-peak child.

5. FareStructure
Table 2.6: Table of the zone coverage and price for each fare type.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InnerZone</td>
<td>int</td>
<td>Primary Key, the innermost zone on the path of the trip, references ZoneID in table “Zones”</td>
</tr>
<tr>
<td>OuterZone</td>
<td>int</td>
<td>Primary Key, the outermost zone on the path of the trip, references ZoneID in table “Zones”</td>
</tr>
<tr>
<td>FareTypeID</td>
<td>int</td>
<td>Primary Key, fare type for the trip, references FareTypeID in table “FareTypes”</td>
</tr>
<tr>
<td>Fare</td>
<td>int</td>
<td>Fare for the trip defined by the inner zone, outer zone, and fare type. Units are US cents</td>
</tr>
</tbody>
</table>

Each fare type has several fare tables with direct/indirect routes and different zone coverage. A direct route is any trip that doesn’t change the direction of travel during the entire trip. An indirect route is any trip that first travels westbound and then changes to eastbound to reach the final destination. Direction change is only allowed from westbound to eastbound service, and the maximum number of direction changes in a single trip is one.

A direct route fare is defined by the innermost zone and outermost zone that the trip includes. A Zone 1-3 direct route fare applies to any trip within zones 1 and 3 using direct routes.

An indirect route fare is defined by the origin zone where the trip starts, the destination zone where the trip ends, and the transfer zone where the direction change happens. A Zone 1-4 via Zone 1 indirect route fare applies to any trip that starts in zone 1 (or 4) traveling westbound, changes to eastbound service in zone 1, and reaches the destination in zone 4 (or 1).

6. ServiceLinks
Table 2.7: Table of train service run between stations.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceID</td>
<td>int</td>
<td>Primary Key, references ServiceID in table “Services”</td>
</tr>
<tr>
<td>OriginStationID</td>
<td>int</td>
<td>Primary Key, the start point station of the service link, references StationID in table “Stations”</td>
</tr>
<tr>
<td>DestStationID</td>
<td>int</td>
<td>Primary Key, the end point station of the service link, references StationID in table “Stations”</td>
</tr>
<tr>
<td>Direction</td>
<td>varchar</td>
<td>Primary Key, the direction of the service link</td>
</tr>
<tr>
<td>Duration</td>
<td>int</td>
<td>Default travel time on this service link. Units are minutes.</td>
</tr>
</tbody>
</table>

7. MorningPeak

Table 2.8: Table of morning peak time period for each station.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StationID</td>
<td>int</td>
<td>Primary Key, station ID, references StationID in table “Stations”</td>
</tr>
<tr>
<td>LastOffPeak</td>
<td>varchar</td>
<td>Last off-peak train before the first peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>FirstPeak</td>
<td>varchar</td>
<td>First peak train after the last off-peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>LastPeak</td>
<td>varchar</td>
<td>Last peak train before the first off-peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>FirstOffPeak</td>
<td>varchar</td>
<td>First off-peak train before the last peak train, in the format of “HH:MM:SS”</td>
</tr>
</tbody>
</table>

8. MorningPeakFri (effective only on Friday)
Table 2.9: Table of morning peak time period on Friday for each station.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StationID</td>
<td>int</td>
<td>Primary Key, station ID, references StationID in table “Stations”</td>
</tr>
<tr>
<td>LastOffPeak</td>
<td>varchar</td>
<td>Last off-peak train before the first peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>FirstPeak</td>
<td>varchar</td>
<td>First peak train after the last off-peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>LastPeak</td>
<td>varchar</td>
<td>Last peak train before the first off-peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>FirstOffPeak</td>
<td>varchar</td>
<td>First off-peak train before the last peak train, in the format of “HH:MM:SS”</td>
</tr>
</tbody>
</table>

9. EveningPeak

Table 2.10: Table of evening peak time period for each station.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StationID</td>
<td>int</td>
<td>Primary Key, station ID, references StationID in table “Stations”</td>
</tr>
<tr>
<td>LastOffPeak</td>
<td>varchar</td>
<td>Last off-peak train before the first peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>FirstPeak</td>
<td>varchar</td>
<td>First peak train after the last off-peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>LastPeak</td>
<td>varchar</td>
<td>Last peak train before the first off-peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>FirstOffPeak</td>
<td>varchar</td>
<td>First off-peak train before the last peak train, in the format of “HH:MM:SS”</td>
</tr>
</tbody>
</table>

10. EveningPeakFri (effective only on Friday)
Table 2.11: Table of evening peak time period on Friday for each station.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StationID</td>
<td>int</td>
<td>Primary Key, station ID, references StationID in table “Stations”</td>
</tr>
<tr>
<td>LastOffPeak</td>
<td>varchar</td>
<td>Last off-peak train before the first peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>FirstPeak</td>
<td>varchar</td>
<td>First peak train after the last off-peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>LastPeak</td>
<td>varchar</td>
<td>Last peak train before the first off-peak train, in the format of “HH:MM:SS”</td>
</tr>
<tr>
<td>FirstOffPeak</td>
<td>varchar</td>
<td>First off-peak train before the last peak train, in the format of “HH:MM:SS”</td>
</tr>
</tbody>
</table>

The morning and evening peak tables, with their Friday variations, are LIRR-specific tables in this design. The LIRR fare policy defines peak and off-peak periods by train, not by passenger boarding or alighting times. This is one of several methods of peak pricing used by commuter railroads; the fare engine implements only this option. The Friday-only tables are used because LIRR schedules are different on Fridays than on other weekdays. All weekend trains are off-peak. The first and last peak train’s times at each station are used to determine whether a tap will be placed in a trip that pays the peak or off-peak fare. Some slack is required, since trains may be late, and passengers arrive and tap in before the train arrives. The detailed strategy for determining peak or off-peak fares is discussed in section 2.5.3.

2.5.2 Building the System Network

In order to generate a path for given tap pairs, a directed graph is built to represent the connections between stations. Stations are represented by nodes while service links are represented by directed arcs between the nodes.
To be able to distinguish different services running between the same station pair and the different directions on the same line or branch, they must be modeled as different services. Separate nodes for each service in each direction must be created to model transfers correctly, as shown in Figure 2-6 (Maciejewski, 2008).

In the network model shown in Figure 2-6, node "1A_E" represents the nodes at Station 1 running Eastbound Service A; solid lines are train services running between stations in different directions; dashed lines are transfer links between different service directions at the same station; and dotted lines are entry and exit links to the station.

Since the LIRR only allows transfers from westbound services to eastbound services at specific stations, only transfer arcs from the westbound direction to the eastbound direction are created when building the network. In this way, the shortest path algorithm will automatically rule out paths with illegal transfers.

The service link between Bethpage on the Ronkonkoma branch and Babylon on
the Babylon branch is eliminated from the network because there are almost no train services on this link, and including it in the network would require complex handling. This issue may be addressed in a more general way in the future.

To distinguish trains running on different branches, each branch is modeled as a separate service. Changing services while traveling in the same direction is not regarded as a transfer in LIRR’s system. Only changing services while changing the direction of travel is regarded as a transfer, and will be recorded by the shortest path algorithm. Some simple examples of transfers in LIRR are given below.

EXAMPLES (refer to Figure 2-7 for the system map):

- **Penn Station → Far Rockaway**
  To complete this trip, the passenger will ride the City Terminal Service from Penn Station to Jamaica, and then change to the Far Rockaway Service at Jamaica to Far Rockaway. There are trains running directly from Penn Station to Far Rockaway, so the passenger is not actually changing trains at Jamaica but only switching service branches. The passenger can also ride a train from Penn Station to Jamaica first, get off the train and ride a second train running from Jamaica to Far Rockaway, in which case the passenger makes a physical transfer at Jamaica. In both cases it is not considered a transfer by the LIRR fare rules, either only changing services or actually changing trains, because both segments of the trip are in the same direction. Direct travel fares will be applied for both cases.

- **Port Washington → Far Rockaway**
  To complete this trip, the passenger will ride the Port Washington Service from Port Washington to Woodside, then change to the City Terminal Service at Woodside to get to Jamaica, and finally change to the Far Rockaway Service at Jamaica to reach Far Rockaway. Because the passenger changes direction of travel from westbound to eastbound at Woodside, this is considered a transfer, while the second service change (from City Terminal to Far Rockaway), as
discussed above, is not considered a transfer. The indirect route “Zone 4-4 via Zone 1” fare will be applied to this trip.

- Penn Station → Atlantic Terminal

To complete this trip, the passenger will ride the eastbound City Terminal Service from Penn Station to Jamaica, and then change to the westbound City Terminal Service at Jamaica to reach Atlantic Terminal. Change of travel direction from eastbound to westbound is not allowed by LIRR, so this trip is considered to be two separate trip segments: Penn Station → Jamaica and Jamaica → Atlantic Terminal, both of which will be charged a Zone 1-3 direct travel fare separately.

Figure 2-7: Schematic of LIRR Services.

2.5.3 Finding the Shortest Path

A shortest path described in Hao and Koucr (1992) and a follow-up paper Dung et al. (1993), is used to get the shortest path for given tap pairs. The shortest path is not necessarily the path with the shortest distance, but can be the path with the lowest fare, shortest travel time, minimum transfer time, etc. The Hao-Kocur algorithm is a variation on the d’Esopo-Pape algorithm, which in turn is a variation on the classic Bellman-Ford algorithm. Hao-Kocur is a dynamic programming algorithm in the
class of so-called “label correcting” algorithms. It is guaranteed to find the optimal path; its worst case running time on a general network is $O(2^n)$, although this is pathological. The worst case running time on a planar network is $O(n \lg(n))$.

There are 3 possible criteria for the shortest path algorithm: minimum fare, minimum travel time and minimum transfers. Minimum travel time or minimum transfers can be computed by keeping track of travel time or transfer times while searching for possible paths. But the minimum fare criterion in LIRR system requires the algorithm to know which type of fare to adopt based on information about the passenger, the current path and time of the day. In some cases, the fare type used in the algorithm may change during the shortest path search. This is because the LIRR has different fares for direct trips and trips with transfers, peak trips and off-peak trips, adults and seniors, etc.

To be able to pick the right fare type, we need to first decide whether the trip is a peak or off-peak trip. Peak and off-peak fares are issued to different groups of people in different time periods. To decide which fare to adopt, we will need such information besides information about the passenger.

**Peak versus Off-Peak Fares**

The LIRR distinguishes peak and off-peak trips by the train that the passenger is riding: it’s a peak trip if the passenger is on a peak train and an off-peak trip otherwise. A peak train is defined by the time the train is scheduled to leave or arrive at New York City, depending on the direction of travel. The peak time band for each station from the LIRR timetable is extracted and stored in the database table. These tables allow the algorithm to decide whether the passenger is on a peak train given the tap-in and tap-out time. A buffer can be applied to allow the passenger some time between tapping and boarding to rule out the possibility of mistakenly charging a peak fare for an off-peak trip.

A trip in the LIRR system can consist of several consecutive rides on different trains. The entire trip is considered as a peak trip if any of these train rides is a peak ride. The algorithm must infer which trains the passenger takes during the trip from
the time of tap-in to the time of tap-out to decide whether the trip qualifies as peak or off-peak. A trip may contain multiple train rides and a trip starting with an off-peak tap-in may be connected with a second ride on a peak train, which makes the entire trip a peak trip. Using both tap-in and tap-out time provides more information about the trip and lowers the chance of mistakenly categorizing a peak trip as an off-peak one.

If the passenger taps in after the departure time of the last off-peak train before the peak period, and it’s earlier than the departure time of the last peak train in the peak period, the passenger is assumed to be riding a peak train. If the passenger taps out after the arrival time of the first peak train in the peak period, and it’s still before the arrival time of the first off-peak train after the peak period, the passenger is assumed to be riding a peak train. The passenger is assumed to be riding an off-peak train if the taps do not meet any of the above conditions. This logic of deciding whether a trip is peak or off-peak is shown in Figure 2-8. The shaded area indicates the peak period in a day.

The standard used by the fare engine to decide whether a trip is peak is:

if (tap-in later than the last off-peak train before the peak period plus a buffer time) AND (tap-in earlier than the last peak train minus a buffer time)
peak = true;
else if (tap-out later than the first peak train plus a buffer time) AND (tap-out earlier than the first off-peak train after the peak period minus a buffer time)
peak = true;
else peak = false;

There are no peak trains on holidays or weekends. Currently the algorithm has
Figure 2-9: Rules used to decide whether a trip is peak or off-peak when there's no off-peak service before the peak period in a day.

an argument indicating whether the date of the tap is a holiday. If true, an off-peak fare is issued for this tap regardless of the time of the tap. The transit agency must provide a table listing all the holidays.

There are some stations in the system which do not have peak (or off-peak) train services or have only one peak train in the morning (or evening) peak. To insure the algorithm will work correctly in deciding whether a trip is a peak one under these special circumstances, certain modifications are made to the peak time table for these stations:

- No off-peak service before the first peak train
  In this case, every passenger who taps in after the last off-peak train from the previous day and before the first peak train of the current day can only ride the peak train. For the algorithm to work, the last off-peak train time should be set as the last off-peak train after the peak period from the previous day, as shown in Figure 2-9.

- No off-peak train after the last peak train
  In this case, every passenger who taps out after the departure time of the last peak train and before the first off-peak train of the next day must have reached the station by the last peak train. For the algorithm to work, the first off-peak train time should be set as the first off-peak train of the next day, as shown in Figure 2-10.

- No off-peak service throughout the day
  This is a combination of the two cases above. In this case, the passenger can
Figure 2-10: Rules used to decide whether a trip is peak or off-peak when there's no off-peak service before the peak period in a day.

only ride a peak train regardless of the time of the tap. For the algorithm to work, the four critical times for the station are set as shown in Table 2.12.

Table 2.12: Critical times for the station when there's no off-peak service.

<table>
<thead>
<tr>
<th>LastOffPeak</th>
<th>FirstPeak</th>
<th>LastPeak</th>
<th>FirstOffPeak</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00:00</td>
<td>00:00:00</td>
<td>23:59:59</td>
<td>23:59:59</td>
</tr>
</tbody>
</table>

- No peak service throughout the day

This is similar to the case above; the passenger can only ride an off-peak train regardless of the time of the tap. For the algorithm to work, the four critical times for the station are set as shown in Table 2.13.

Table 2.13: Critical times for the station when there's no peak service.

<table>
<thead>
<tr>
<th>LastOffPeak</th>
<th>FirstPeak</th>
<th>LastPeak</th>
<th>FirstOffPeak</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00:00</td>
<td>00:00:00</td>
<td>00:00:00</td>
<td>00:00:00</td>
</tr>
</tbody>
</table>

Discount Groups and Fare Types

The fare engine handles three different passenger groups in the LIRR system: adult, senior/student/disabled, and children traveling alone. These are the basic passenger types in the LIRR system.
In the current implementation, each phone is only allowed to pay for a single passenger. Using the same phone to pay for multiple passengers is not supported. One possible future feature is to let the passenger indicate how many passengers are paid for by this phone for the specified trip.

After deciding whether the trip is peak or not, the appropriate fare type for this trip can be chosen according to the passenger’s group type and the time the trip is taken.

1. Morning Peak: Peak fare for adults, seniors/disabled and child.

2. Evening Peak: Peak fare for adults; half of peak fare for seniors/disabled and child.

3. Off-Peak: Off-peak fare for adults; half of peak fare for seniors/disabled; half of off-peak fare for child.

This fare system requires the fare engine to know the path between the station pairs, to be able to tell the direction of travel. Getting the path requires a known fare type to be used in the shortest path algorithm. In the current implementation, peak fares are used by default to first get the path and then the model re-calculates the fare based on the path and passenger information. It is theoretically possible that there may be another path under a different fare type which has a cheaper fare in some transit systems. However, in the LIRR network there is only one path between all station pairs because it has a tree structure, and thus the fare type will not change the possible paths.

Fares for bus and subways are currently set to be a single fare of \$2.25. In the real system, there are some exceptions in bus fares, which are not covered in this research.

A penalty of \$20 is applied for penalized taps. This parameter, and the bus and subway fares, can be configured to any desired value.

**Invalid Tap Pairs**

Invalid taps should be filtered out as early as possible on the tap server. Invalid taps are taps with invalid fields such as invalid station id, discount group, time format,
gtfs_type, etc. The trip server and fare engine server must also have the ability to handle invalid inputs, since there may be errors that they can detect while the tap server cannot.

If the time gap between a tap-in and a tap-out is larger than the maximum allowed travel time in the system, the fare engine will return an error message. Currently the maximum allowed travel time is set to 3 hours. This threshold can be set to different values for each station pair if more information on the default travel times between stations is available in the future.

1. **Invalid Station ID**: The fare engine will map the station id field to that in the database. If it cannot find a matching station id in the database, it will return an error message to the trip server and the trip server will mark the tap as an “INVALID_TAP” with price $0.00.

2. **Invalid Time**: Taps with an invalid time format will be considered as a peak trip.

3. **Invalid Discount Group**: A trip with an invalid discount group will be considered “ADULT”, which is the most expensive fare among all groups.

4. **Invalid gtfs_type**: This is handled by the trip server. Such taps will be ignored and marked as an “INVALID_TAP” with price $0.00.

**Best Price Guarantee**

The current implementation guarantees the best fare for every single trip, but the best price guarantee for a span of time is left out for now. It can be added to the system by going over all the trips throughout a time period. The best price guarantee will require that we have all the tap sequences for a single client for the period over which we guarantee the best price, such as a week, 10 days, or a month.

Additional fare types that are considered in the current model are:

1. **Monthly and Weekly Fares**: For adults and children above 5. This type of commutation ticket allows direct routes only, but if both segments of a trip are
within a zone limit, then it may also apply to this trip by considering the two segments as individual trips.  

**Example:** Consider a trip from Port Washington to Far Rockaway. This is a Zone 4 to Zone 4 via Zone 1 indirect trip. But we can assume the passenger first makes a trip from Port Washington to Woodside and then makes a second trip from Woodside to Far Rockaway, both of which are a Zone 1 to Zone 4 direct trip. In this case both segments of the trip fall into the zone limit of a Zone 1-4 Monthly Ticket, so the ticket can be applied to both segments of the trip although this trip is an indirect trip.

2. **Extension of Ride:** Extension is allowed in conjunction with commutation tickets, providing the full one-way fare is collected for travel between the zone limit allowed on the commutation ticket to the extended zone requested by the customer.  

**Simple example:** If a passenger holds a monthly pass for Zone 4-7, and the trip is from zone 9 to zone 4, then the passenger should pay an extension fare of a one-way zone 7-9 fare.  

**Complex example:** If a passenger holds a monthly pass for Zone 4-7, and the trip is from zone 9 to zone 7 via zone 3, then the passenger should pay an extension fare of a one-way zone 7-9 fare and twice the one-way zone 3-7 fare.

3. **Ten-trip:** These are valid over a direct route only to or from zone 1. If an off-peak ten-trip is presented on a peak train (direct route only), one tenth (1/10) of the purchase price of the ten-trip off-peak ticket will be applied toward the one-way peak fare and the difference, rounded up to the next higher $.25 increment, (step-up) collected. In this way the cost for an off-peak trip from/to zone 1 with a ten trip pass is 1/10 of the purchase price. In fact the ten-trip peak ticket is exactly 10 times of the one-way peak fare. So there’s no need to implement a best price guarantee for ten-trip peak ticket with current pricing.

Two fare types are not considered in the current model:
1. **Monthly School Pass**: This type of ticket has very strict application rules, which are not considered for now.

2. **Family Fare and Group Fare**: These are not included for now because paying for a group with the same phone is not allowed for now.

### 2.5.4 Request and Response Data

The request string from the trip server to fare engine is formatted as a JSONArray shown as follows. The JSONArray consists of 2 taps. The data below is an example only.

```json
{
  "tapsGrouped": [
    {
      "id": "789",
      "t_time": "2012-06-25 09:07:16",
      "u_id": "352425050850851",
      "t_type": "1",
      "card_r": "3834306472386d682d",
      "event_type": "TAP_IN",
      "discount_group": "ADULT",
      "trip_id": "789#790",
    },
    {
      "id": "790",
      "t_time": "2012-06-25 09:48:22",
      "u_id": "352425050850851",
      "t_type": "1",
      "card_r": "3834306472386d682d",
    }
  ]
}
```

49
"event_type": "TAP_OUT",
"discount_group": "ADULT",
"trip_id": "789#790",
}
}

The response string from the fare engine to the trip server is formatted as a JSONArray shown as follows. The response is either an error message or a JSONArray of the priced trips.

"Invalid tap pair"

"Origin station invalid"

"Destination station invalid"

{"tapsPriced": [  

{

  "price": "875",
  "trip_id": "789#790",

}

]}

}
Chapter 3

Performance

Currently the trip server and fare engine are organized in a way that they can handle multiple requests (multiple threads) at the same time. For simplicity, after sending requests (tap server to trip server, or trip server to fare engine server) the trip server or fare engine server will block the port so that no other request can be passed through until the response is sent back. This can be easily solved by separating the request and response part, e.g., having the request and response assigned to different ports. For the current pilot project, a single thread between servers is sufficient.

Using the system and algorithm design specified above, the system ran for a 6-month pilot. It correctly grouped taps into trip segments and calculated the correct fare while handling duplicate and missing taps. This system design and pilot project proved the feasibility of a mobile ticketing system.

Tap data collected from volunteers in the pilot is used to illustrate the system output, as shown in Table 3.1. These taps are all from a single passenger account. For clarity, some fields in the data are eliminated.
Table 3.1: Critical times for the station when there’s no peak service.

<table>
<thead>
<tr>
<th>TapID</th>
<th>Station</th>
<th>Event</th>
<th>Price</th>
<th>Trip_ID</th>
<th>Tap_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5095</td>
<td>Port Washington</td>
<td>PENALIZED_TAP</td>
<td>2000</td>
<td>5095</td>
<td>2012-05-30 16:44:00.0</td>
</tr>
<tr>
<td>5515</td>
<td>Penn Station</td>
<td>TAP_IN</td>
<td>500</td>
<td>5515#5516#5519#5520#5521</td>
<td>2012-06-01 20:38:00.0</td>
</tr>
<tr>
<td>5516</td>
<td>Penn Station</td>
<td>TAP_IN</td>
<td>500</td>
<td>5515#5516#5519#5520#5521</td>
<td>2012-06-01 20:38:00.0</td>
</tr>
<tr>
<td>5519</td>
<td>Penn Station</td>
<td>TAP.OUT</td>
<td>500</td>
<td>5515#5516#5519#5520#5521</td>
<td>2012-06-01 21:12:00.0</td>
</tr>
<tr>
<td>5520</td>
<td>Penn Station</td>
<td>TAP.OUT</td>
<td>500</td>
<td>5515#5516#5519#5520#5521</td>
<td>2012-06-01 21:12:00.0</td>
</tr>
<tr>
<td>5521</td>
<td>Penn Station</td>
<td>TAP.OUT</td>
<td>500</td>
<td>5515#5516#5519#5520#5521</td>
<td>2012-06-01 21:12:00.0</td>
</tr>
<tr>
<td>5522</td>
<td>Penn Station</td>
<td>PENALIZED_TAP</td>
<td>2000</td>
<td>5522</td>
<td>2012-06-01 21:13:00.0</td>
</tr>
<tr>
<td>5649</td>
<td>Murray Hill</td>
<td>TAP_IN</td>
<td>875</td>
<td>5649#5683</td>
<td>2012-06-04 08:36:00.0</td>
</tr>
<tr>
<td>5683</td>
<td>Penn Station</td>
<td>TAP.OUT</td>
<td>875</td>
<td>5649#5683</td>
<td>2012-06-04 09:03:00.0</td>
</tr>
<tr>
<td>5829</td>
<td>Penn Station</td>
<td>TAP_IN</td>
<td>875</td>
<td>5829#5847</td>
<td>2012-06-04 19:03:00.0</td>
</tr>
<tr>
<td>5847</td>
<td>Murray Hill</td>
<td>TAP.OUT</td>
<td>875</td>
<td>5829#5847</td>
<td>2012-06-04 19:36:00.0</td>
</tr>
</tbody>
</table>
There is a two-day time gap between the first tap, #5095, and the second tap, #5515. This time gap exceeds the allowed maximum travel time, so the first tap is considered to belong to an incomplete trip and is penalized $20.

Taps #5515 and #5516 are at the same station and the time gap between these two taps is very small, so they are considered duplicate tap-ins. The same rule is applied to taps #5519, #5520, #5521, and they are all considered duplicate tap-outs at the same station. A zone 1-1 off-peak fare of $5.00 is charged. All the taps-in and taps-out are at Penn Station, so the passenger may have not made any train trips. This data is from the beginning of the pilot and the participants may have been testing the devices, which produced these duplicate taps. (The algorithm can be further improved to include actual travel time between stations. If the time between tap-in and tap-out at the same station is shorter than the fastest system travel time, then the passenger can be considered as not making any trip and will not be charged for these taps. This can occur if, for example, the passenger taps in but then realizes he or she forgot something at home or at the office, and returns to the station later for actual travel.)

Tap #5522 is penalized for the same reason as tap #5095. It is assumed to be a tap-in without a corresponding tap out. Tap #5649 and #5683 are paired as a tap-in and tap-out of a morning peak trip from Murray Hill to Penn Station, which is charged a zone 1-3 direct route peak fare of $8.75.

Tap #5829 and #5847 are paired as a tap-in and tap-out of an evening peak trip from Penn Station to Murray Hill, which is charged a zone 1-3 direct route peak fare of $8.75.
Chapter 4

GTFS Extension

The system described in Chapter 2 is specifically designed for LIRR’s network and fare structure, but a good open payment system should be compatible with multiple transit systems, and should provide easy transitions from one system to another.

4.1 GTFS Background

The General Transit Feed Specification (GTFS), developed by Google, defines a common format for public transportation schedules and associated geographic information (Google, 2014d). GTFS “feeds” allow public transit agencies to publish their transit data and developers to write applications that consume that data in an interoperable way.

A GTFS feed is composed of a series of text files collected in a ZIP file. Each file models a particular aspect of transit information: stops, routes, trips, fares and other schedule data.

4.2 Existing Problem and Suggested Modification

Making the fare engine compatible with data in GTFS format will potentially allow a single mobile ticketing system to be used on many transit systems. The current GTFS format, although it provides a good standard for publishing time schedules and
Table 4.1: GTFS Feed File Specifications. (Google, 2014b)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Required</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>agency.txt</td>
<td>Required</td>
<td>One or more transit agencies that provide the data in this feed.</td>
</tr>
<tr>
<td>stops.txt</td>
<td>Required</td>
<td>Individual locations where vehicles pick up or drop off passengers.</td>
</tr>
<tr>
<td>routes.txt</td>
<td>Required</td>
<td>Transit routes. A route is a group of trips that are displayed to riders as a single service.</td>
</tr>
<tr>
<td>trips.txt</td>
<td>Required</td>
<td>Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.</td>
</tr>
<tr>
<td>stop_times.txt</td>
<td>Required</td>
<td>Times that a vehicle arrives at and departs from individual stops for each trip.</td>
</tr>
<tr>
<td>calendar.txt</td>
<td>Required</td>
<td>Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.</td>
</tr>
<tr>
<td>calendar.dates.txt</td>
<td>Optional</td>
<td>Exceptions for the service IDs defined in the calendar.txt file. If calendar.dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt.</td>
</tr>
<tr>
<td>fare_attributes.txt</td>
<td>Optional</td>
<td>Fare information for a transit organization’s routes.</td>
</tr>
<tr>
<td>fare_rules.txt</td>
<td>Optional</td>
<td>Rules for applying fare information for a transit organization’s routes.</td>
</tr>
<tr>
<td>shapes.txt</td>
<td>Optional</td>
<td>Rules for drawing lines on a map to represent a transit organization’s routes.</td>
</tr>
<tr>
<td>frequencies.txt</td>
<td>Optional</td>
<td>Headway (time between trips) for routes with variable frequency of service.</td>
</tr>
<tr>
<td>transfers.txt</td>
<td>Optional</td>
<td>Rules for making connections at transfer points between routes.</td>
</tr>
<tr>
<td>feed_info.txt</td>
<td>Optional</td>
<td>Additional information about the feed itself, including publisher, version, and expiration information.</td>
</tr>
</tbody>
</table>
geographic information, is very limited in describing fare structures. It only covers
peak adult fares which are either zone based or flat. Most transit systems have more
complex fare structures that can't be modeled in GTFS.

Modifications to the fare_rule.txt, fare_attribute.txt and other files in GTFS must
be made before a general mobile ticketing fare engine can be developed that uses
GTFS data.

4.2.1 Peak versus Off-peak Service

GTFS only covers peak fares, but many transit systems have different fares for peak
and off-peak services. For example, in the LIRR system, peak and off-peak fares
are offered depending on the arrival time at New York City (morning peak) or the
departure time from New York City (evening peak). To be able to distinguish fares for
peak and off-peak trips, information on whether the trip qualifies as peak or off-peak
should be provided.

The current trips.txt file in GTFS describes trips (in LIRR’s case, trains) running
on each route, but it provides no information on whether the trip is peak or off-peak,
as shown in Table 4.2 (Google, 2014e).

In order to model the difference between peak and off-peak service, the trip.txt
file in GTFS can be extended to include peak and off-peak information for each train,
for systems such as LIRR that define peak and off-peak fares train by train. Other
systems will define peak versus off-peak fares based on time of entry (or possibly exit)
from the system; this must be handled differently, as described later in extensions
made to fare rule specifications.
Table 4.2: Field Definitions, Trip File.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Required</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>route_id</td>
<td>Required</td>
<td>The route_id field contains an ID that uniquely identifies a route. This value is referenced from the routes.txt file.</td>
</tr>
<tr>
<td>service_id</td>
<td>Required</td>
<td>The service_id contains an ID that uniquely identifies a set of dates when service is available for one or more routes. This value is referenced from the calendar.txt or calendar_dates.txt file.</td>
</tr>
<tr>
<td>trip_id</td>
<td>Required</td>
<td>The trip_id field contains an ID that identifies a trip. The trip_id is dataset unique.</td>
</tr>
<tr>
<td>trip_headsign</td>
<td>Optional</td>
<td>The trip_headsign field contains the text that appears on a sign that identifies the trip's destination to passengers. Use this field to distinguish between different patterns of service in the same route. If the headsign changes during a trip, you can override the trip_headsign by specifying values for the the stop_headsign field in stop_times.txt.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.2 – continued from previous page

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Required</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>trip_short_name</td>
<td>Optional</td>
<td>The trip_short_name field contains the text that appears in schedules and sign boards to identify the trip to passengers, for example, to identify train numbers for commuter rail trips. If riders do not commonly rely on trip names, please leave this field blank. A trip_short_name value, if provided, should uniquely identify a trip within a service day; it should not be used for destination names or limited/express designations.</td>
</tr>
<tr>
<td>direction_id</td>
<td>Optional</td>
<td>The direction_id field contains a binary value that indicates the direction of travel for a trip. Use this field to distinguish between bi-directional trips with the same route_id. This field is not used in routing; it provides a way to separate trips by direction when publishing time tables. You can specify names for each direction with the trip_headsign field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - travel in one direction (e.g. outbound travel)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - travel in the opposite direction (e.g. inbound travel)</td>
</tr>
</tbody>
</table>
Table 4.2 – continued from previous page

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Required</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>block.id</td>
<td>Optional</td>
<td>The block.id field identifies the block to which the trip belongs. A block consists of two or more sequential trips made using the same vehicle, where a passenger can transfer from one trip to the next just by staying in the vehicle. The block.id must be referenced by two or more trips in trips.txt.</td>
</tr>
<tr>
<td>shape.id</td>
<td>Optional</td>
<td>The shape.id field contains an ID that defines a shape for the trip. This value is referenced from the shapes.txt file. The shapes.txt file allows you to define how a line should be drawn on the map to represent a trip.</td>
</tr>
</tbody>
</table>
| wheelchair_accessible | Optional | Integer value  
  - 0 (or empty) - indicates that there is no accessibility information for the trip  
  - 1 - indicates that the vehicle being used on this particular trip can accommodate at least one rider in a wheelchair  
  - 2 - indicates that no riders in wheelchairs can be accommodated on this trip  |

Continued on next page
Table 4.2 – continued from previous page

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Required</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_period</td>
<td>Optional (New)</td>
<td>The time_period field identifies the time period of the day/week that this trip is running. This field can be used by fare_rules.txt file to specify the system fare structure. Suggested values are: peak, off-peak, weekend, night, etc.</td>
</tr>
<tr>
<td>service_name</td>
<td>Optional (New)</td>
<td>The service_name field identifies the service this trip belongs to. This field can be used by fare_rules.txt file to specify the system fare structure.</td>
</tr>
</tbody>
</table>
Table 4.3: Example of trip.txt file of LIRR with “service_type” field.

<table>
<thead>
<tr>
<th>route_id</th>
<th>service_id</th>
<th>trip_id</th>
<th>direction_id</th>
<th>time_period(new)</th>
<th>service_name (new)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Off-peak</td>
<td>City Terminal</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>40</td>
<td>1</td>
<td>Peak</td>
<td>City Terminal</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>604</td>
<td>0</td>
<td>Off-peak</td>
<td>Port Washington</td>
</tr>
</tbody>
</table>

A new field called “time_period”, indicating the time period of the day/week when the trip is running must be added to the trips.txt file. This field can be used to provide information on whether the trip is peak or off-peak. Suggested values of this field are “peak”, “off-peak”, “weekend”, “night”, etc.

In addition to information on whether the trip is peak or off-peak, a “service_name” field should also be added to provide information on which service/branch this trip belongs to, which will be used by fare_rules.txt to model the system fare structure.

EXAMPLE

Table 4.3 shows a sample data of trip file in GTFS formate of LIRR with new fields added.

Trip #1 and trip #40 are two trains running eastbound on the City Terminal Branch at different times of the day. Without the “time_period” field, trips made on these two trains with the same origin and destination station pairs show no difference in fares because both trips run on the same route and have the same zone range. With “time_period” added, these two trips now belong to different service types (peak and off-peak), which can be used to calculate the fare, which will be discussed later in the fare rules section.

Direction and service branch information is also included in new fields because the LIRR has transfer rules that affect the fare. Only transfers from westbound service (into New York) to eastbound service (away from New York) are allowed on some branches. Without “service_name” and “direction” fields, transfers made between different directions and branches would be considered to be the same when calculating
the fare. With the new fields, fare rules can be specified on transfer directions and change of service, which will be discussed later in the fare rules section.

4.2.2 Fare Rules Specification

The fare-rules.txt file in GTFS specifies the rules that must be met when applying a fare to the trip. Detailed field specifications are shown in Table 4.4 (Google, 2014a). The current GTFS specification can only cover fare structures which are either zone based or route based; this extension allows a broader set of fare policies.

To be able to model both peak and off-peak fares with complex transfer rules, four new fields are added to fare-rules.txt: “from-service”, “to-service”, “transfers”, and “time_period”. “from_service” and “to_service” are referenced from “service_name” in the trips.txt file. These two fields will specify all transfers between different service types that are allowed by each fare and “transfer” will specify the maximum number of transfers allowed for each type of transfer. The “to_service” field can be left empty to model fares that apply only to direct trips of the service types specified in “from_service”. The “transfers” field, when left empty, indicates unlimited transfers between services. Modeling more complex fare structures is possible with information provided by “time_period” field in trips.txt (peak, off-peak, weekend, night, etc.). An extreme case is to define every single trip as a separate service type and then the fare rules can be specified in detail for each trip.

EXAMPLE

Assume there’s a fare type that applies to off-peak trips between zone 1 and zone 3 on the City Terminal Branch and Port Washington Branch that do not change direction, or change from Port Washington Westbound (toward New York) to City Terminal Eastbound (away from New York) only once.

Using the current fare rules specification in GTFS, this fare can be described as (in the order of “fare_id, origin_id, destination_id, contains_id”):

1, 1, 3, 1

63
1, 1, 3, 3

The specification above will allow this fare to be applied to any peak or off-peak trip between zone 1 and zone 3 on all branches with unlimited transfers, which is not the actual fare rule.

Using the revised fare rules specification in GTFS, this fare can be described as (in the order of “fare_id, origin_id, destination_id, contains_id, from_direction_id, from_time_period, from_service, to_direction_id, to_time_period, to_service, transfers”):

1, 1, 3, 1/3, 1, OffPeak, City Terminal, , ,
1, 1, 3, 1/3, 0, OffPeak, City Terminal, , ,
1, 1, 3, 1/3, 1, OffPeak, Port Washington, , ,
1, 1, 3, 1/3, 0, OffPeak, Port Washington, , ,
1, 1, 3, 1/3, 1, OffPeak, City Terminal, 1, OffPeak, Port Washington,
1, 1, 3, 1/3, 0, OffPeak, City Terminal, 0, OffPeak, Port Washington,
1, 1, 3, 1/3, 0, OffPeak, Port Washington Westbound, 1, OffPeak, City Terminal Eastbound, 1

With the revised specification, this fare no longer applies to any peak trips, trips on branches other than City Terminal or Port Washington, trips transferring from eastbound to westbound service, or trips transferring from City Terminal westbound to Port Washington eastbound.
Table 4.4: Field Definitions, Fare Rule File.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Required</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>fare_id</td>
<td>Required</td>
<td>The fare_id field contains an ID that uniquely identifies a fare class. This value is referenced from the fare_attributes.txt file.</td>
</tr>
<tr>
<td>route_id</td>
<td>Optional</td>
<td>The route_id field associates the fare ID with a route. Route IDs are referenced from the routes.txt file. If you have several routes with the same fare attributes, create a row in fare_rules.txt for each route.</td>
</tr>
<tr>
<td>origin_id</td>
<td>Optional</td>
<td>The origin_id field associates the fare ID with an origin zone ID. Zone IDs are referenced from the stops.txt file. If you have several origin IDs with the same fare attributes, create a row in fare_rules.txt for each origin ID.</td>
</tr>
<tr>
<td>destination_id</td>
<td>Optional</td>
<td>The destination_id field associates the fare ID with a destination zone ID. Zone IDs are referenced from the stops.txt file. If you have several destination IDs with the same fare attributes, create a row in fare_rules.txt for each destination ID.</td>
</tr>
<tr>
<td>contains_id</td>
<td>Optional</td>
<td>The contains_id field associates the fare ID with a zone ID, referenced from the stops.txt file. The fare ID is then associated with itineraries that pass through every contains_id zone.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.4 – continued from previous page

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Required</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>from_service</td>
<td>Optional (New)</td>
<td>The from_service field refers to the service_name field in trips.txt file. Along with to_service field, it specifies the transfers between services that are covered by the fare.</td>
</tr>
<tr>
<td>to_service</td>
<td>Optional (New)</td>
<td>The to_service field refers to the service_name field in trips.txt file. Along with from_service field, it specifies the transfers between services that are covered by the fare. When left empty, the fare can be applied to direct trips of the type specified in from_service field.</td>
</tr>
<tr>
<td>transfers</td>
<td>Optional (New)</td>
<td>The transfers field specifies the allowed maximum number of transfers specified by from_service and to_service fields.</td>
</tr>
<tr>
<td>time_period</td>
<td>Optional (New)</td>
<td>The time_period field refers to the time_period field in trips.txt file. It specifies the time period that this fare can be applied.</td>
</tr>
</tbody>
</table>
With these proposed changes, the entire fare structure of LIRR can be modeled correctly in GTFS. LIRR has a time sensitive zone-based fare with extended transfer rules. With time_period field added to the trips.txt file, fares can be specified in more detail, specifying which trips during which time period are covered. With the service_name field added, more information on trips is provided to be used by the fare rules to specify service coverage and transfer rules for each fare class.

While the LIRR defines peak and off-peak period by the operating time of trains, many other transit systems, such as Transport for London (TfL), define peak and off-peak periods by the time when the passenger enters/exits the system. In order to model such fare structures using GTFS, new fields need to be introduced to the fare rules file besides the ones discussed above. The following discussion about fare rule extensions is based on the TfL system.

TfL has a zone-based fare structure which is defined by the inner-most zone and outer-most zone traveled along the entire trip, regardless of where the trip starts or ends. It also provides peak and off-peak fares based on the tap in time rather than the actual time when the service leaves. In TfL system, every journey has its own maximum journey time, regardless of the route taken. This is linked to the day, time of day travelled and the number of zone boundaries crossed.

To model the tap time sensitive, zone-based fares using the GTFS fare_rules.txt, origin_id and destination_id can be left empty because the TfL fare structure is insensitive to where the trip starts or ends. The contains zone should include all the zones between the inner-most and outer-most zones so that any trip that travel through these zones are covered by this fare. A zone 1-4 fare in TfL system can be written as “, , 1/2/3/4, …” in GTFS format.

TfL distinguishes peak and off-peak services by the tap in time, not the time of the service that the passenger is taking. New field names “tap_in_from, tap_in_to, tap_out_from, tap_out_to” are added to the fare_rules.txt to model this kind of fare structure. Trips with tap in times within the range defined by tap_in_from and tap_in_to together can be covered by this fare. Similarly, fare rules can also specify tap out time range to define peak and off-peak services when needed.
To model the maximum journey time in TfL, a new field “duration” is added to fare_rules.txt. TfL’s maximum journey times vary among trips made in different zone ranges and number of zones crossed along the journey. These limits on maximum journey time will bring a lot of complexity to the GTFS models, which is not common in other transit systems. So here the maximum journey time is set to a fixed value based on zone range, regardless of number of zones crossed along the journey. Transit agencies may provide their own add-on algorithm besides the GTFS standard if they want to consider these special requirements. The maximum journey time also varies across days in a week, so a new field “calendar_id” is introduced to indicate days in a week when the fare can be applied. Transit agencies are responsible of providing such information when building GTFS data.

**EXAMPLE**

The TfL’s “Tube, DLR and London Overground Adult Zone 1-2 Peak Single” fare is £2.80. Peak is defined as tapping in the range of 06:30-09:30 or 16:00-19:00 Monday to Friday. The maximum journey time within zone 1 and 2 is 100 minutes during Mon-Fri 04:30-19:00. So the GTFS fare rules for this fare can be written as shown in the following table. Note that two fare classes are created for this single fare because it applies to 2 different time ranges in a single day.

The proposed new structure for fare rule files provides the ability to model fare structures that fall into one or more of the following categories: (1) time sensitive; (2) zone-based; (3) route-based; (4) flat; (5) with transfer restrictions that can be expressed in a fromService-toService table.

### 4.2.3 Modeling of other MTA fares Bus and Subway

The fare for a subway or local bus ride is $2.25 and the fare for an express bus ride is $5.50 in the MTA system. This can be modeled by specifying subway, local bus and express bus as having separate service_name values.

A fare of $2.25 can be specified to cover direct trips of subway or local bus services,
Table 4.5: Example of TfL’s fare rule in extended GTFS format.

<table>
<thead>
<tr>
<th>Column</th>
<th>Fare Rule Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fare_id</td>
<td>1</td>
</tr>
<tr>
<td>Route_id</td>
<td>(empty)</td>
</tr>
<tr>
<td>Origin_id</td>
<td>(empty)</td>
</tr>
<tr>
<td>Destination_id</td>
<td>(empty)</td>
</tr>
<tr>
<td>Contains_id</td>
<td>1/2</td>
</tr>
<tr>
<td>From_service</td>
<td>Tube, DLR and London</td>
</tr>
<tr>
<td></td>
<td>Overground</td>
</tr>
<tr>
<td>From_direction</td>
<td>(empty)</td>
</tr>
<tr>
<td>From_time_period</td>
<td>(empty)</td>
</tr>
<tr>
<td>To_service</td>
<td>(empty)</td>
</tr>
<tr>
<td>To_direction</td>
<td>(empty)</td>
</tr>
<tr>
<td>To_time_period</td>
<td>(empty)</td>
</tr>
<tr>
<td>Tap_in_from</td>
<td>06:30</td>
</tr>
<tr>
<td>Tap_in_to</td>
<td>09:30</td>
</tr>
<tr>
<td>Tap_out_from</td>
<td>(empty)</td>
</tr>
<tr>
<td>Tap_out_to</td>
<td>(empty)</td>
</tr>
<tr>
<td>Duration</td>
<td>100 (unit: minute)</td>
</tr>
<tr>
<td>Calendar_id</td>
<td>(this field should refer to the service_id in calendar.txt)</td>
</tr>
</tbody>
</table>

69
and one free transfer from subway service to local bus service but not vice versa. The free bus-to-bus transfer can also be specified using the from_service/to_service field. This will require the service_name of each bus to be specified in more detail to include route and direction information. A fare of $5.50 can be specified as only covering direct trips of express bus services.

4.3 Network Model and Fare Calculation

The shortest path algorithm in the fare engine must be modified to be able to work with GTFS data with the proposed new fields. Also, a network model is created on which the shortest path algorithm is run to find the lowest fares (the “shortest path” is the “least fare” path). The network construction, which is done automatically based on the revised GTFS files, is described first, and then the shortest path algorithm is also described.

4.3.1 Building the Network to Support Fare Calculations

Using the new fields in the GTFS data, the stop_times.txt file specifying actual train schedules is used to construct the network (Google, 2014c). With this change, a more detailed, train by train, network can be built. A node is created for each entry in the stop_times.txt file, and entry (I, for ingress) and exit (E, for exit) nodes are created for each station, as shown in Figure 4-1.

Service “a” runs on route 4-2-3; service “b” runs on route 2-3; service “c” runs on route 2-3-5. Each node represents a train stop at a certain station with arrival and departure times (showing one time only in Figure 4-1 for clarity). Solid lines are train service links between stop nodes, dotted lines are walking links to enter or exit a station, and dashed lines are walking links to transfer between services within a station.

Stop nodes at the same station are sorted by arrival time. Transfer links are created automatically between every pair of consecutive stop nodes.

A transfer link is also created from the node with the latest arrival time to the node
Figure 4-1: Network structure built from stop_times.txt.

with the earliest arrival time at the same station. This represents all transfers that are physically possible for passengers, regardless of whether the transfer is allowed by the system’s fare structure. A node with a later arrival time can reach a node with an earlier departure time through these transfer links and this is considered an overnight transfer. Overnight transfer links are included in the network because a passenger may arrive at the station late in the night at 11:50pm, and the next train is not until 00:15am the next day. In this case, the passenger have to stay overnight for the next train. the network is constructed to only include stop nodes in a single day from 00:00:00 to 23:59:59, so the calculation of transfer times over these overnight transfer links will be taken care of by the shortest path algorithm. Transfers that are not covered by the fare structure will be treated as ending the previous trip and starting a new one, which will be charged as two separate trips. This will be discussed later in the section 4.3.4.

4.3.2 Shortest Path Algorithm for Direct Routes

The network is built to include any physically possible links between stop nodes. However, there may be routes between station pairs in the network that are not
covered by a single fare type. The modified shortest path algorithm is designed to find only routes that can be covered by a single fare. Segmentation of trips that must pay two or more fares is handled in a separate method, which is discussed in section 4.3.4.

In order to get the correct fare, a list of all the service names and transfers made along the trip is kept in the label for each node. This list is compared to the fare rules’ “from/to service” list. A fare type which can cover all the services and transfers in the node’s label is applied. If there is more than one class of applicable fare, all of them are kept in different labels for the node. This is because even some of the fare class may offer a higher fare at the current node, as the algorithm proceeds to new nodes towards the destination, the now more expensive fare class may turn out to be the cheapest that can be applied to the entire path.

**EXAMPLE**

This example in Figure 4-2 is used to demonstrate the different result between the two strategies of (1) only keeping the cheapest fare class that can be applied to the path, and (2) keeping labels of all applicable fare classes for a single path.

Suppose we're looking for path from node 2 to node 4. A path is 2-1-5-4 and the other path is 2-3-5-4. Node 2 and node 5 are in zone 2, node 1 is in zone 1, node 3 is in zone 3, and node 4 is in zone 4. There are 3 classes of fares: zone 1-3, zone 2-4 and zone 1-4; the price is in ascending order.
1. Only keeping the cheapest fare class that can be applied to the path

When the algorithm reaches node 5, both path 2-1-5 and path 2-3-5 will have a label of the zone 1-3 fare. Then suppose path 2-1-5 has shorter travel time, so it's kept in the label set and path 2-3-5 is deleted. But when the algorithm reaches node 4, only the zone 1-4 fare is applicable because path 2-3-5 has been deleted. But in fact 2-3-5-4 is the cheapest path overall.

2. Keeping labels of all applicable fare classes for a single path

When the algorithm reaches node 5, both path 2-1-5 and path 2-3-5 will have a label of the zone 1-3 and zone 1-4 fare, while path 2-3-5 will also have a label of the zone 2-4 fare. Then suppose path 2-1-5 has the shorter travel time, so its zone 1-3 label is kept and the zone 1-3 label for path 2-3-5 is deleted. Later when the algorithm reaches node 4, only the zone 1-4 fare is applicable to path 2-1-5-4 while the zone 2-4 fare is applicable to path 2-3-5-4, which is the cheapest among all the labels.

To keep the number of labels in the node's label set from exploding combinatorially, the applied fare class is also kept in the label. After updating a node's label, the label set of the node is updated to eliminate dominated labels of the same fare class. This strategy is based on the fact that if two labels have the same applicable fare class, they have the same potential zone and service coverage the services and zones traveled along both of the paths to get to the current node are already covered by this fare class and any connecting path from this point that is also covered by the fare class is shared by these two paths, so there's no difference between these two paths when applying this specific fare class.

In order to eliminate the labels with the same fare class for each node, the concept of composite cost is introduced. Composite cost is defined as the total cost, both money and time, the passenger encounters during the trip, which is the sum of the fare and the equivalent money value of the time spent on this trip (VOT×travel_time). Here VOT stands for “Value of Time”, which is the maximum money a passenger is
willing to pay to save a unit of time in his/her trip. VOT can be personalized for each customer to reveal different travel preferences. Different VOTs for waiting time, transfer time, in-vehicle time can also be set to get a more detailed user preference model.

Each label has a field of “composite_cost” and among all the labels with the same fare class, the one with the lowest composite cost (travel time and fare cost) is kept. In this way, the maximum number of labels for a single node can be no more than the number of total fare classes in the system.

The travel time between any two directly connected nodes is calculated by subtracting the arrival time of the previous node from the arrival time of the latter node. Because the network is constructed to include any physically possible transfer links, there may be cases when the arrival time of the latter node is earlier than the previous node. Such travel between the nodes is an overnight transfer, and the travel time is automatically increased by 24 hours. This guarantees that the travel time is always increasing as the algorithm proceeds to new nodes.

EXAMPLE

Suppose there are trains running from station A to station B every 30 minutes and it takes 15 minutes for the train to reach B from A. Any train leaving station A between 7:00-10:00 AM is a peak train and the user is charged a peak fare of $5.00. Trains running at any other time are off-peak and the fare is $3.00.

The VOT of the passenger is set as $0.02/min, which means the passenger is willing to pay $0.02 to save a minute in his/her travel time.

If the passenger arrives at station A at 9:40 AM, he/she can catch the 10:00 AM peak train or he/she can wait a little longer to catch the 10:30 AM off-peak train. The composite cost of the first option is 5.00 + 0.02×(20+15) = $5.70 while the composite cost of the second option is 3.00 + 0.02×(50+15) = $4.30. So waiting for the off-peak train is preferred over riding the peak train.

If the passenger arrives at station A at 7:40 AM, he/she can catch the 8:00 AM peak train or he/she can wait longer to catch the 10:30 AM off-peak train. The
composite cost of the first option is $5.00 + 0.02 \times (20+15) = $5.70 while the composite cost of the second option is $3.00 + 0.02 \times (170+15) = $6.70. So riding the peak train is preferred over waiting for the off-peak train.

But using VOT may sometimes yield unreasonable results (e.g. the algorithm may suggest the passenger wait at the platform for 1 hour to catch an off-peak train instead of jumping onto the peak train leaving in 2 minutes), mostly due to the range of pre-set VOT. To avoid such cases, the algorithm is set to only consider the trips leaving the station in the next T minutes after the passenger arrives at any certain station the station can either be the origin station or any intermediate transfer station along the trip. This T is called the maximum waiting time and a reasonable range of T would be 10-30 minutes.

4.3.3 Shortest Path Algorithm with Dwell Times at Stations

The basic algorithm described above works for a network in which the arrival time and departure time are the same for each stop node (i.e., the train arrives at 8:05 and also departs at 8:05). When the arrival time differs from the departure time at a station, which occurs at large stations or transfer points, this way of constructing the network and getting the transfer time no longer works.

Taking the network shown in the Figure 4-3 as an example, stop nodes for station 2 are created with arrival time and departure time, sorted in ascending order of arrival time. Suppose a passenger arrives at station 2 by the 08:02:00 train at node 2. He or she can catch the train departing at 08:05:00 from node 1 to reach his or her final destination. The actual transfer time from node 2 to node 1 is 3 minutes. But in the simple network construction method described in section 4.3.1, the transfer route from node 2 to node 1 is 2-3-...-n-1, and the transfer is considered an overnight transfer which takes 23h58min.

The network should instead be constructed by directly connecting any two stop nodes at the same station where direct transfer is physically possible, as shown in Figure 4-4. The transfer time should be calculated by subtracting the arrival time of the previous node from the departure time of the later node. The travel time of the
Figure 4-3: Sample Network: stop nodes connected sequentially by arrival time.

Figure 4-4: Sample Network: stop nodes connected to each other at the same station.

service links between stations should be the difference between the departure time of the previous node and the arrival time of the later node.

The transfer time between stop nodes at the same station is calculated by subtracting the arrival time of precious stop node from the departure time of the connecting stop node. This strategy will introduce errors as going back in time as described below:

Assume the three stop nodes in the Table 4.6 all belong to the same station and a passenger arrives at the station on a train at node 1. Since the departure time of node 2 is later than the arrival time of node 1, the passenger can transfer from node 1 to node 2 with a 1 minute transfer time. Now the passenger is at node 2 and the

Table 4.6: Example: arrival and departure time at a station.

<table>
<thead>
<tr>
<th>Node Number</th>
<th>Arrival Time</th>
<th>Departure Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>08:00</td>
<td>08:05</td>
</tr>
<tr>
<td>2</td>
<td>07:56</td>
<td>08:01</td>
</tr>
<tr>
<td>3</td>
<td>07:53</td>
<td>07:58</td>
</tr>
</tbody>
</table>
departure time of node 3 is later than the arrival time of node 2, the algorithm will allow the passenger to transfer from node 2 to node 3 with a 2 minute transfer. After the second transfer, the passenger has transferred from node 1 to node 3 with total transfer time of 3 minutes, which in fact should be an overnight transfer of 23h58min.

To solve this problem, arrival node and departure node of the same stop node should be modeled separately, as demonstrated in the following figure. Dashed lines are transfer links within the same station while solid lines are service links connecting the stop nodes of the same trip (train). Transfer links should only be created from arrival node to departure node within the same station and service links should be created from arrival node to departure node of the same trip stop. In this way, two or more consecutive transfers are not allowed within the same station and going back in time is no longer possible. Any transfer with a departure time earlier than the arrival time is considered as an overnight transfer and the transfer time is the actual time it will cost the passenger to take the transfer link.

4.3.4 Trip with Change of Fare Class

Because the network is built to include any physically possible connection between stations, regardless of the rules of travel in the system, there may be routes that can’t be covered by a single fare class. The shortest path algorithm is configured to
only take care of routes that can be covered by a single fare class. Such routes may also include legal transfers that are covered by certain fare types. Segmentation of trips with change of fare class is taken care of by a separate method discussed in this section. The change of fare class happens at the break point that divides the trip into separate trip segments that can each be covered by a single fare class. In the fare engine, the minimum number of such changes for a given origin-destination pair is decided using an adjacency matrix prior to the choice of strategy and then the algorithm will choose the segmentation strategy accordingly.

An adjacency matrix is used to represent the pairs of stations that can be reached using a single fare type. By calling the shortest path algorithm from every station as the origin, the adjacency matrix $A$ of the network can be derived by setting $a_{ij}$ equal to 1 if there exists a trip between station $i$ and $j$ that can be covered by a single fare type, and 0 otherwise. Then by self-multiplying this adjacency matrix $A$, the non-zero entries in matrix $A^2$ show all station pairs that can be connected with 0 or 1 “change of fare class”. Station pairs that can be connected with 2 and 3 “transfers” are computed in a similar way with $A^3$ and $A^4$. Trips with more than 3 “transfers” are not considered in the fare engine because seldom will anyone make that many transfers in a single journey, but the fare engine can be configured to allow any number of transfers.

Based on the number of “transfers” (changes of fare class) needed for any station pair, as obtained from the adjacency matrices, segmentation strategies are described as follows:

1. For an OD pair that can be connected with ONLY 1 “transfer”.
   - Pick a station that can be reached from the origin station and that can reach the destination station using a single fare type (pick station $t$ if $a_{ot} > 0$ and $a_{td} > 0$, where $o$ stands for origin, $d$ stands for destination, $t$ stands for transfer station);
   - Call the shortest path algorithm with the origin station set as station $t$ and the destination station set as $d$. Pass the label set of station $t$ from the
previous trip segment origin-transfer as an argument to be the initial label set of the root node for the second call to the shortest path algorithm;

- Pick the label from the label set of the destination station with the lowest composite cost as the best route for origin-transfer-destination;

- Repeat the above 3 steps for all transfer stations and pick the one with the lowest composite cost at the destination station as the best route.

2. For an origin-destination pair that can be connected with ONLY 2 “change of fare class”.

- Pick any stations $t_1$ and $t_2$ that qualify: $a_{ot_1} > 0$, $a_{t_1t_2} > 0$ and $a_{t_2d} > 0$. This means there exists a trip with one fare class between the origin and $t_1$, $t_1$ and $t_2$, $t_2$ and the destination.

- Call the shortest path algorithm with origin station set as station $t_1$ and destination station set as $t_2$, passing the label set of station $t_1$ from previous trip segment origin-$t_1$ as an argument;

- Call the shortest path algorithm with origin station set as station $t_2$ and destination station set as $d$, passing the label set of station $t_2$ from previous trip segment $t_1$-$t_2$ as an argument;

- Pick the label from the label set of the destination station with the lowest composite cost as the best route for origin-$t_1$-$t_2$-destination;

- Repeat the above 4 steps for all transfer station pairs $(t_1, t_2)$ and pick the one with the lowest composite cost at the destination station as the best route.

3. For any OD pair than can be connected with ONLY 3 “change of fare class”

- Pick any stations $t_1$, $t_2$ and $t_3$ that qualify $a_{ot_1} > 0$, $a_{t_1t_2} > 0$, $a_{t_2t_3} > 0$ and $a_{t_3d} > 0$. This means there exists single trip between origin and $t_1$, $t_1$ and $t_2$, $t_2$ and $t_3$, $t_3$ and destination. The rest steps are the obvious extensions of the first two cases.
EXAMPLE

Nodes in the above figure are stations and directed lines are services running between the station nodes. Each line style stands for a separate service line and is covered by a flat fare. No transfers between different service lines are allowed in the system. For example, a trip from node 2 to node 3 on solid line is charged $1 while a trip from node 3 to node 6 on the express service line (dashed line) is charged $2. A trip from node 2 to node 6 includes a transfer of service at node 3, and will be considered as two separate trips (trip 2-3 on solid line and trip 3-6 on dashed line). The price for this trip is $3 ($1 for the first trip and $2 for the second).

A passenger taps into the system at node 1 and taps out at node 8. The algorithm should find how much the passenger should be charged under the assumption that he/she takes the cheapest path.

The adjacency matrix \( A \) of the sample network is shown in (4.1).

With \( a_{18} = 0 \), it’s clear there’s no direct path from node 1 to node 8 and a change of fare class must be applied to this trip. By self-multiplying this adjacency matrix \( A \), \( a_{18}^2 = 1 \) indicates that node 8 can be reached from node 1 with only 1 “transfer” (change of fare class). We find all the node indexes \( t \) in \( A \) so that \( a_{1t} = 1 \) and \( a_{t8} = 1 \): nodes 3 and 5 are the two indexes that satisfy this condition.

The shortest path algorithm is first called to find the paths between node 1 and node 3, and then the label set of node 3 from this trip segment is passed as an argument to the second call of the shortest path algorithm to find paths between node 3 and node 8. The path with the lowest composite cost in node 8’s label set is
chosen as the candidate path. Trip segment 1-5 and 5-8 is processed in a similar way and another path candidate is chosen from node 8’s label set.

Finally these two candidate paths are compared and the one with smaller composite cost is chosen and returned.

$$
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
2 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
3 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
4 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
$$

(4.1)

4.4 Further Extensions to Other Transit System

This section gives a brief review of several metropolitan transit systems’ fare structure and how to model them using the proposed GTFS extension.

1. Beijing

- Fare Structure: Beijing offers a flat fare of CNY 2 for unlimited transfers within the subway system. Bus fares in Beijing are either flat or distance based. The distance based fare has a station-station fare table for each bus route.

- Modeling Approach: Flat fares are covered by existing GTFS. For bus fares, zone numbers can be assigned to each individual bus station, and fares can be specified using the current design of from.zone/to.zone fields.

2. Shanghai
• Fare Structure: Shanghai offers flat/distance-based fares for both of its subway and bus system.

• Modeling Approach: Similar to Beijing using from_zone/to_zone fields.

3. Sydney

• Fare Structure: Sydney offers section-based fares for buses—each service line has its own sections (similar to zones), and fares are based on how many sections are traveled on the trip. Sydney’s subway system has a distance-based fare structure with certain restrictions on circumferential routes.

• Modeling Approach: Section-based fares for buses can be modeled as zone-based fares while distance-based subway fares can be handled in a similar way as for Beijing and Shanghai.

4. Boston

• Fare Structure: Boston offers flat/zone-based fare for both subways and buses with one free transfer between subway-bus/bus-bus within 2 hours.

• Modeling Approach: Flat and zone-based fares are covered by the proposed GTFS extension with maximum travel durations.

5. Singapore

• Fare Structure: Singapore offers distance-based fares for both subways and buses with a maximum of 5 transfers and one entry/exit of the subway network for each journey which is only valid for 2 hours. A rider can use a specific bus line in a single journey only once. The fare is calculated for the exact distance the passenger has traveled using a lookup table. Singapore also offers special discount and free ride to adult and senior passengers based on the tap-out time and destination station.

• Modeling Approach: The current algorithm keeps the distance traveled on each route in the label. Fares can be easily calculated by looking at the
distance-fare table. For the special discount and free ride, new fare class with discounted fare or zero fare can be created and the tap.out.from and tap.out.to in the extended GTFS can set to the time range when such discount is offered. The zone range can be set to constrain the usage of such discounted fare to certain stations.

6. San Francisco

- Fare Structure: Muni offers a flat fare with unlimited transfers and a maximum journey time of 90 minutes; BART offers a distance-based fare with a station-station fare table; AC Transit offers a flat fare with an extra transfer fee and a maximum journey time of 2 hours.

- Modeling Approach: Flat fares are covered by the proposed GTFS extension while distance-based fares can be modeled in a similar way as for Beijing and Shanghai by assigning a zone number to each individual station.
Chapter 5

Conclusion and Future Work

Although it is not known how soon transit agencies may accept mobile NFC payments from the public, this research has proved the feasibility of a future NFC mobile ticketing system.

In this research, the system requirements of a NFC mobile ticketing system for public transit services are defined as to be able to collect taps at ticketing gates, verify the validity of the taps, group the taps into trip segments according to system-specific rules, and finally calculate the fare for each trip the user makes and charge it to the user’s account.

Based on the system requirements, a system design is proposed for a NFC mobile ticketing pilot: the system’s function is divided into three different servers with separate fundamental functions to allow flexibility in a Public-Private Partnership. The tap server serves as the system’s interface that directly communicates with the user (user’s phone) so that the entire ticketing process appears to be executed within a single request from the passenger’s point of view. The tap server’s function is designed to collect, store and validate all taps sent from users’ NFC enabled phones. The tap server is also responsible for initiating the trip segmentation of received taps. It has been shown that such design and implementation of a tap server is capable of collecting and validating all the taps generated during the pilot, while successfully serving as the communication channel between the ticketing system and the user’s phone.
The research focuses on the design and implementation of trip server and fare engine in the system, which should be system-specific to be able to accommodate the different network structures and service rules in various transit systems, while the rest of the system can be shared across multiple transit agencies or outsourced to vendors who serve non-transit merchants. The current design has left out the actual billing of the trip to the user’s account. The next step of the NFC pilot deployment should include the billing of priced trips to the user’s bank account to make sure that the technology is completely functional when introduced to the public market. A billing engine should be designed and implemented with the participation of other players in the open payment system.

The function of grouping taps into trip segments is implemented as a finite state machine on the trip server for LIRR rail trips and MTA bus and subway trips. The trip server successfully segmented taps collected for selected users into trip segments, according to LIRR’s service and transfer rules. Exception handling such as duplicate taps at the same station due to reader failure or customers’ unfamiliarity to the system is handled by the trip server to avoid overcharging the customer’s account. Missing taps are considered as potential fraud and penalized by the trip server.

Only taps at LIRR ticket gates, fare boxes on MTA buses and MTA subway gates are included in the current design for simplicity. Future extensions to the trip server should also include the taps at intermediate validators and on-board inspection devices for completeness. This will introduce more states into the state machine and extra exception handling should also be considered. The fare engine can be modified to work with multiple types of taps on the same trip. Free transfers between buses and subways are another direction of future extension to the trip server. Time gaps between subway and bus taps and the distance between the transferring stations are the crucial criteria for the qualification of the free transfer.

The fare engine described in Chapter 2 is specifically designed for LIRR’s network and fare structure. The fare engine can find the path and fare for different user groups under different criteria (minimum fare/travel time/transfer counts) between LIRR station pairs while satisfying the system’s service rules, such as restrictions on
transfers. The fare engine was tested on the pilot data and successfully returned the correct fares for direct trips, trips with transfers, and multiple trips priced separately between station pairs that can’t be reached directly under a single fare class. Trips in the pilot are post-paid and the fare calculation is run on the fare engine server instead of on the user’s phone, which removes the computation burden from the phone.

The design of the fare engine has left out some functions for simplicity. Default travel time between station pairs can be added to the algorithm so that trips which take significantly shorter or longer travel time than the default value can be identified as being potential fraudulent. A more advanced improvement could compare the actual travel time between taps to the system’s real time schedule between station pairs for more accurate fraud identification. The best price guarantee over time periods is another key element of the mobile ticketing system and should be included in a future release of the pilot. This will require the implementation of group and period (monthly, weekly) fare ticketing in the fare engine. Personalized travel preferences, customized VOT, etc. can be added to the fare engine to enhance the performance of the system.

A general fare engine that can serve multiple transit systems will greatly improve the ease of deployment of mobile ticketing in the future: there’s no need to develop separate fare algorithm for each individual transit system and the fare engine is easy to upgrade/maintain once there’s any change in the transit system service/fare structures.

The combination of GTFS extensions, the network generation method and the modified shortest path algorithm proposed in Chapter 4 forms the basis for a general fare calculation application that could be used on servers or clients (desktops or mobile phones) to compute fares—as well as travel times and transfers—for users and transit systems, and to provide the pricing mechanism for future open payment systems that might be based on tapping NFC phones at gates and fare boxes.

The proposed approach was tested on LIRR’s and TfL’s networks under various scenarios: direct trips, trips with transfers, trips with boundary stations, peak and off-peak trips, etc., all of which returned the correct fare within seconds. Fare structures
from other major transit agencies should be modeled and tested using the proposed GTFS extension to evaluate its potential as a general transit service and fare structure feed specification standard.

This pilot is a good example that NFC mobile ticketing can be of great practical use. NFC is very intuitive and easy to use: the data transmission requires no more than a simple touch, just like the existing contactless card technologies used in public transit payment. The volunteers in the pilot learned how to use the NFC enabled phone to tap at gates very quickly. The acceptance rates are very high, with the participants mainly citing convenience. They would no longer have to obtain a new fare card when traveling to a new city nor would they need to reload fare cards or wait in line to purchase tickets in busy periods. Further benefits can be the integration of location based services and other value-added services. For example, when traveling on an unfamiliar route, a location based service can notify the user when to get off the bus and where to transfer to the subway.

The benefits transit agencies receive from mobile ticketing include cost savings, reduced staff count, marketing, sales and distribution, and enhanced security. It will also provide opportunities for transit agencies to access new channels via the mobile handset to bring value added services such as traveller guides and service alerts to passengers.
Bibliography

  URL: http://gfuse.com/blog/london-buses-and-nfc-tags/


  URL: http://www.capmetro.org/app/

  URL: http://www.chilternrailways.co.uk/content/using-app

  URL: http://www.dart.org/gopass/


  URL: https://developers.google.com/transit/gtfs/reference#fare_rules_fields

  URL: https://developers.google.com/transit/gtfs/reference

  URL: https://developers.google.com/transit/gtfs/reference#stop_times_fields

  URL: https://developers.google.com/transit/gtfs/

  URL: https://developers.google.com/transit/gtfs/reference#trips_fields


89
Juniper Research (2012). mTicket mTransport mEvent, Whitepaper by Juniper Research.


National Retail Federation (2011). Mobile retailing blueprint: A comprehensive guide for navigating the mobile landscape, National Retail Federation.


Smart Card Alliance (2012). Near field communication (nfc) and transit: Applications, technology and implementation considerations, Smart Card Alliance, Princeton Junction, NJ.
URL: http://www.totalrail.org/2014/03/12/barcode-mobile-ticketing-gates-mass-transport-uk-lead-way

URL: http://trimet.org/mobiletickets/index.htm