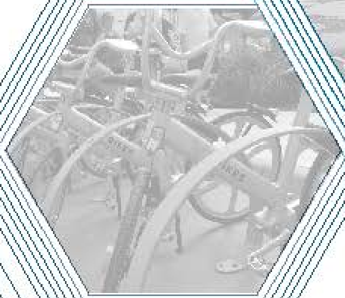


IMPROVING TRANSPORTATION PERFORMANCE MEASUREMENT VIA OPEN "BIG DATA" SYSTEMS: PHASE 1

Principal Investigators:
Sean J. Barbeau, Ph.D.
Robert L. Bertini, Ph.D.

FINAL REPORT

Center for Transportation Equity, Decisions & Dollars

IMPROVING TRANSPORTATION PERFORMANCE MEASUREMENT VIA OPEN "BIG DATA" SYSTEMS
PHASE 1 TRANSIT

FINAL PROJECT REPORT

by

Sean J. Barbeau, Ph.D.
Minh Pham, M.A.
Jorge Adorno Nieves, M.S.
Robert L. Bertini, Ph.D.
University of South Florida

for

Center for Transportation, Equity, Decisions and Dollars (**CTEDD)**
USDOT University Transportation Center
The University of Texas at Arlington
601 W.Nedderman Dr. Suite 103
Arlington TX 76019-0108 United States
Phone: 817-272-5138 | Email: C-Tedd@uta.edu

In cooperation with US Department of Transportation-Research and Innovative Technology Administration (RITA)

Acknowledgment

This work was supported by a grant from the Center for Transportation Equity, Decisions and Dollars (CTEDD) funded by U.S. Department of Transportation Research and Innovative Technology Administration (OST-R) and housed at The University of Texas at Arlington.

Disclaimer

Technical Report Documentation Page

1. Report No.	**2. Government Accession No.**	**3. Recipient's Catalog No.**	
4. Title and Subtitle Improving Transportation Performance Measurement via Open 'Big Data' Systems – Phase 1 Transit		**5. Report Date**	
		6. Performing Organization Code	
7. Author(s) Sean J. Barbeau, Ph.D., Minh Pham, M.A., Jorge Adorno Nieves, M.S., Robert Bertini, Ph.D.		**8. Performing Organization Report No.**	
9. Performing Organization Name and Address Center for Transportation, Equity, Decisions and Dollars (CTEDD) USDOT University Transportation Center The University of Texas at Arlington 601 W.Nedderman Dr. Suite 103 Arlington TX 76019-0108 United States		**10. Work Unit No. (TRAIS)**	
		11. Contract or Grant No.	
12. Sponsoring Organization Name and Address United States of America Department of Transportation Research and Innovative Technology Administration		**13. Type of Report and Period Covered**	
		14. Sponsoring Agency Code	
15. Supplementary Notes Report uploaded at www.ctedd.uta.edu			
16. Abstract Within public transportation, data-driven metrics are fundamental to an agency's ability to properly plan and manage the resources within their network. Past efforts to analyze the performance of these systems have been hampered by a lack of centralized real-time data archives. Issues such as difficult data acquisition, varying data formats, and limited transferability and generalization prevent new tools from being deployed at other transit agencies. This project's goal is to lay the foundation for a "big data" centralized repository supporting the dynamic, ongoing archival of real-time multimodal information to assist practitioners, researchers, and students in better understanding the transportation network across multiple regional geographic areas. We introduce a novel system for archiving, retrieval, and use of real-time and scheduled public transit data, which can serve as a foundation for performance assessment, big-data analysis and machine learning applications. By leveraging standardized data formats and new software technologies, these tasks can be performed across multiple agencies concurrently. Archiving multiple transit agencies simultaneously allows researchers to compare approaches with different agencies without needing to alter their approaches for a specific dataset. Deployment metrics after eight months of data archiving and estimated deployment costs in the long term are presented for the initial seven testing datasets. The system is made available as an open-source software project to encourage active collaboration with other practitioners and operations personnel. Further collaborations between researchers and analysts is supported via centralized infrastructure where researchers can retrieve archived data and corroborate techniques across several datasets. Doing so will directly contribute to measuring the efficiency and quality of public transportation. In addition, we demonstrate how existing tools for on-time performance measurement and accessibility analysis can be modified to use archived real-time data from the system. Results from these analyses could be leveraged by transit agencies, researchers, and metropolitan planning organizations to better understand the difference between scheduled and actual transit service and how this impacts riders and the public transportation system.			
17. Key Words big data, real-time transit data, performance measurement, GTFS, GTFS-realtime		**18. Distribution Statement** No restrictions.	
19. Security Classification (of this report) Unclassified.	**20. Security Classification (of this page)** Unclassified.	**21. No. of Pages** 37	**22. Price** NA

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Management guru Peter Drucker is quoted as saying “What gets measured gets managed.” This principle has driven some of the early efforts to archive transportation data on a regional basis. One example of such a database is Portland State University’s PORTAL system, which houses data from the Portland Oregon and Vancouver, Washington metro areas [1]. PORTAL contains freeway speeds and volumes, arterial counts, arterial signal cycle data, Bluetooth arterial travel times, transit ridership and on-time performance, and bicycle and pedestrian counts. Past studies have also applied this data-driven performance management concept to transit service, with the measurements based on archived vehicle tracking data [2, 3]. The archived vehicle tracking data give a picture of how the transit service is truly operating, not what it should be doing (i.e., the schedule). However, there are several key limitations to the current state of the art for the empirical evaluation of transit service via vehicle tracking data including difficult data acquisition, varying data formats, limited transferability/generalization of analysis, a limited capacity of the agencies for processing these data themselves, as well as variation of parameters used to determine on-time performance [4]:

1. Difficult data acquisition – Vehicle tracking data typically comes directly from the transit agency and is a static copy of the historical tracking databases from the agency’s automatic vehicle location (AVL) system. Since these data are maintained within the agency, planners and analysts must request access and the agency must manually export and transport the data to them. Exporting and transporting the data can be difficult, since these datasets are extremely large and cannot be transported over simple electronic tools such as email.
2. Varying data formats – Because vehicle tracking systems are proprietary, agencies’ vehicle tracking databases are formatted differently depending on their vendors. This leads to significant pre-processing effort required by planners and analysts before any analysis can begin, which increases the cost of measuring on-time performance to improve operations.
3. Limited transferability/generalization – Because of the effort involved in acquisition and preprocessing of vehicle tracking datasets, it is difficult for the transit agency and state and regional planners to produce an analysis that covers more than one transit agency’s data and allows systems of different sizes to benchmark performance to peers in other states. As a result, it is unknown whether any analysis or techniques for improving transit service are generalizable across multiple agencies.
4. Limited capacity of the agencies for processing these data themselves – While transit agencies often have access to these data internally, they seldom have the skill set or time necessary to turn substantial amounts of data into actionable information. As a result, opportunities for a variety of improvements (e.g., validating/improving arrival predictions for rider-facing information, improving schedules and routes) are lost.

Creating an ongoing centralized archive of real transit network behavior based on open-source software, including archived public transportation vehicle data in an easily accessible standardized format, can address these challenges. Such an architecture will directly contribute to measuring the efficiency and quality of transportation as well as ensuring transportation system vitality through performance management and monitoring systems. Using this centralized system, the efficiency and quality of transit service can easily be analyzed and compared across cities by developing techniques that are generalizable across the datasets. The open-source nature of the software facilitates collaboration with other researchers and analysts. Analysts can spend their time analyzing the already-collected datasets rather than trying to find and reformat various proprietary data dumps (e.g., for transit, from Computer Aided Dispatch / Automatic Vehicle Location (CAD/AVL) systems). Easier analysis will lead to better and more consistent performance metrics, both within the same local systems over several years as well as across different transit agencies/cities. This same stream of real-time and archived data can also serve as input to monitoring systems, which can also be applied across multiple agencies and cities.

The objective of this project is to facilitate collaborative archiving of multimodal transportation data sources, which will be used to fuel enhancements based on “big data”-driven transportation performance measures. Performance measures are important in public transportation because data-driven improvements can reduce carbon dioxide emissions, ensure a region’s accessibility coverage, and maintain the transportation system in a good state. This phase of the project focuses on the archival of real-time transit information.

We have utilized an agile software engineering methodology to implement open-source software that can scale to archiving many different transit feeds from around the world on an ongoing basis. We have collected data that includes vehicle positions (including latitude, longitude, speed, heading, and occupancy if available), estimated arrival times (including uncertainty values if available), and service alerts from agency GTFS-realtime feeds. In addition, we have also successfully modified open-source analytical tools so that they can work with the collected data. These tools include Retro-GTFS [5] for on-time performance analysis and Conveyal Analysis [6] for accessibility analysis. This work implies that the process and system developed for this project is capable of fueling many types of research, performance measures, and metropolitan planning activities by providing rich and abundant real-time transit data.

The remainder of this document is organized as follows. Chapter 2 describes the methodology for the transit data archival system along with its deployment metrics and estimated costs. Chapter 3 describes how the Retro-GTFS tools was integrated into our system to perform on-time performance analysis. Chapter 4 describes how the Conveyal Analysis tool was modified to accepts data from our system to perform accessibility analysis.

2. SCALABLE REAL-TIME TRANSIT DATA ARCHIVING

2.1 DESCRIPTION OF TRANSIT DATA

Since 2010, the General Transit Feed Specification (GTFS) [7] has emerged as a dominant format for transit schedule data, typically used to power mobile apps [8]. This has allowed analysis of schedule data over a large number of agencies via the collection of GTFS data from centralized forums such as the GTFS Data Exchange [7], TransitFeeds.com [9], and Transitland [10]. In the last few years, a real-time counterpart to GTFS, GTFS-realtime [11], has begun to emerge, with many agencies sharing their real-time data in this format with mobile apps. However, no centralized repository for archived GTFS-realtime feeds has emerged.

A GTFS dataset, or feed, is a collection of six required text files and seven optional text files. Together, they form a relational database that describes a transit system's scheduled operations. The six required text files are:

- agency.txt: Provides information about the transit agency, including name, website, time zone, and contact information.
- routes.txt: Identifies distinct routes, each with a route ID and descriptive information.
- trips.txt: Identifies trips, their associated route ID, and service ID. Service ID can be used to join trips.txt with calendar.txt to identify the trip's service days.
- stop.txt: Defines the geographic location of all actual stops or stations in the transit system.
- stop_times.txt: Provides scheduled arrival and departure time for each stop of each trip.
- calendar.txt: Defines whether the services operate on each weekday and/or weekend, including service reoccurrences. Each service pattern is associated with a service ID.

Seven additional files are optional to the specification: calendar_dates.txt, fare_attributes.txt, fare_rules.txt, shapes.txt, frequencies.txt, transfers.txt, feed_info.txt. shapes.txt, while optional, is important, as it provides the actual travel path of the vehicle.

A GTFS-realtime (GTFS-RT) feed is a protocol buffer feed that consists of three types of entities:

- Trip Updates: Represents changes in the schedule. These updates give predicted arrival or departure time for stops along the operating trips. Trip updates can also provide information for scenarios where trips are canceled.
- Vehicle Position: Provide information on the locations of an agencies' vehicles (e.g., GPS coordinates).
- Service Alerts: Provide human-readable descriptions for disruptions on the network for reasons such as weather, medical emergency, technical problem, or holiday. Delays and

cancellations of individual trips should be communicated through Trip Updates instead of Service Alerts.

2.2 DESIGN OF THE DATA ARCHIVAL SYSTEM

The architecture of the system is divided into several components within a two-tier hierarchy: core components and application components (Figure 1). Core Components relate to the archive and access of archived transit data while Application Components include utilities that are able to retrieve archived information in real time. This allows the Core Components to act as a framework for data retrieval to the remaining applications and abstract information such as where the data is coming from, how it is stored and whether it is valid or not.

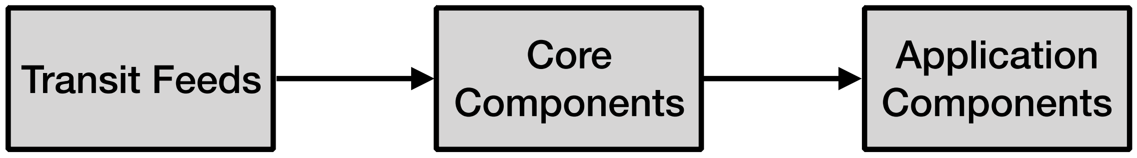


Figure 1: Diagram of System Architecture

2.2.1 Core Components

The architecture of the Core Components is displayed in Figure 2.

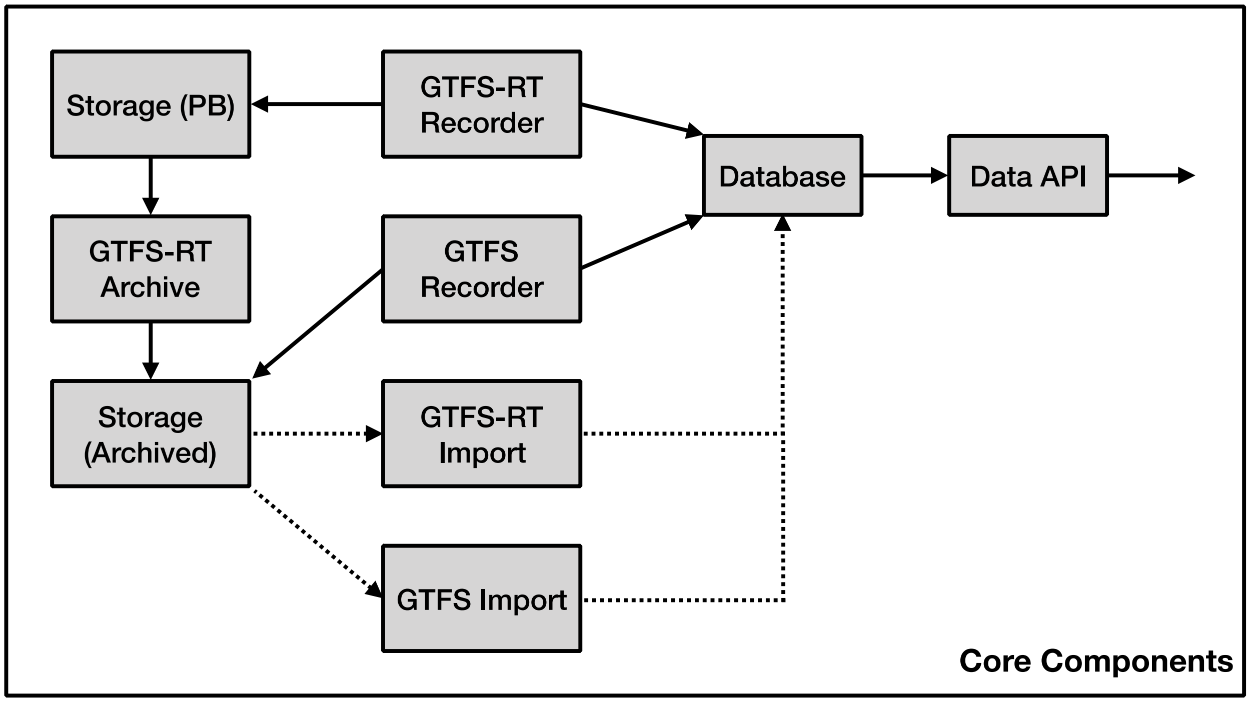


Figure 2: Diagram of Core components

Each of the components is described in the following section from a single agency's perspective. Support for archiving multiple agencies is discussed in section 2.3.

- **GTFS-RT Recorder**: The GTFS-RT Recorder implements a module that decodes the vehicle and trip protocol buffers and stores each as a single JavaScript Object Notation (JSON) document in a NO-SQL Database. This allows all data fields used by the transit agency to be stored, which enables the database to easily reject duplicate entries and pushes the responsibility for managing missing fields to another component down the data pipeline. The GTFS-RT Recorder also exports the original protocol buffer data into the local file system, which is the "Storage (PB)" module below, so an original unaltered version of the data retrieved from the transit agency exists. The GTFS-RT Archive component then prepares this data for final long-term storage in the "Storage (Archived)" module. This protocol buffer data can also be imported into a new database implementation via the GTFS-RT Import module if another analyst or researcher wants to examine data on their own system. Being one of the most critical components of the system, the GTFS Recorder supports flexible configuration via runtime parameters including the feed URLs and target database.
- **GTFS Recorder**: A separate tool to retrieve, archive, and import GTFS information, modeled after the GTFS-RT Recorder. Although all of the data is formatted in a tabular

structure, the GTFS Recorder component converts each of the files inside a GTFS archive into their respective JSON document without paying special attention as to which files and columns an agency has made available. Exceptions are made for trips.txt, stop_times.txt and shapes.txt, which, due to their size, are imported as multiple documents grouped around the file's route_id, trip_id, and shape_id columns. Each of these entries is tagged with a timestamp related to the GTFS archive, which allows duplicates to be automatically rejected by the database and allows iteration in a historical manner. Lastly, GTFS zip files fetched from transit agencies that are not duplicates of previous data are stored in its original state for future use or distribution.

- **Database**: This component stores GTFS and GTFS-RT data as retrievable entries across various tables. Each table is dedicated to a single file or feed retrieved through the recorders. By design no stateful data is stored here. This allows the Data API to handle requests for any point in time and allows a database to be fully restored from the "Storage (Archived)" module via the GTFS and GTFS-RT Importers. A non-relational database is essential to the implementation of the system. If relational database was used, developers would need to choose between allocating columns on relational databases for all the possible fields or selectively choosing fields to allocate from the GTFS-RT specification. A similar situation occurs when data is archived over time. As transit agencies update the data they offer through their feeds, non-relational databases are able to add this information without performing changes to the application performing archiving or the database structure. Relational implementations would require both the tool and database to be modified to add new fields.
- **Storage (PB)**: The distribution of GTFS data is relatively simple due to the required delivery method being a single compressed zip file and only being updated around 3-4 times a year by most agencies. GTFS-RT data is very different, as the protocol buffer files provided by agencies are much smaller but updated constantly (e.g., every 15 seconds). As a result, storing the many raw individual protocol buffer files can take up significant disk space. A solution to this problem is to store protocol buffers in a temporary location after they are downloaded and routinely package them together into a permanent archive optimized for storage space. The Storage (PB) module serves as a temporary storage of the raw protocol buffer GTFS-RT files downloaded from agencies. These are converted into a compressed long-term format, the "Storage (Archived)" module, by the GTFS-RT Archiver module.
- **GTFS-RT Archiver**: The GTFS-RT Archiver converts the raw protocol buffer files from the Storage (PB) module into a compressed permanent storage of the data in the Storage (Archived) module. This component is scheduled to run periodically and produces shareable archives that contain the original, unaltered data provided by a transit agency.
- **Storage (Archived)**: This module contains the GTFS-RT protocol buffer data archived from agencies in a format optimized for disk space and distribution. It can be used to initialize a new Database with the archived data via the GTFS-RT Importer module.

- **GTFS-RT Importer**: Complementing the components previously mentioned, the GTFS-RT Importer is able to read compressed GTFS-RT archives from the "Storage (Archived)" module and import them into the Database module. Shown in Figure 2 as the dotted lines, this component allows the configuration of a new system which previously contained no archived data. This is useful for researchers that would like to get started analyzing available GTFS-RT datasets retrieved from the centralized "Storage (Archived)" model on their own systems without needing to go through the process of archiving the data themselves. This component is also capable of importing extensions to the GTFS-RT format for data fields specific to a particular agency that are not yet adopted into the GTFS-RT specification, and could integrate supplementary proprietary data formats with GTFS-RT data if needed.
- **GTFS Importer**: Similar to the GTFS-RT Recorder, this component imports GTFS archived into the database. Refer to the GTFS Recorder discussion for a more detailed explanation on how data is organized within the database.
- **Data Application Programming Interface (API)**: Providing direct access to a database usually poses security risks to the data stored and increases the complexity of a tool's desired tasks. For this reason, an HTTP-based Application Programming Interface (API) is implemented. The Data API fetches the GTFS and GTFS-RT data from their corresponding tables and delivers it to the requester, handles errors relating to queries to the database, verifies that results from GTFS tables are valid for a given timestamp, and delivers special variables to other components of the system such as feed URLs, time zones, and transit agency name. In addition to serving information to other components, this is one of the externally facing components that can provide historical data stored in the database in a fast and efficient manner to external entities (e.g., other researchers).

2.2.2 Application Components

The Application Components of the system are shown in Figure 3. These applications assist researchers and analysts in accomplishing specific tasks, using data provided by the Core Components.

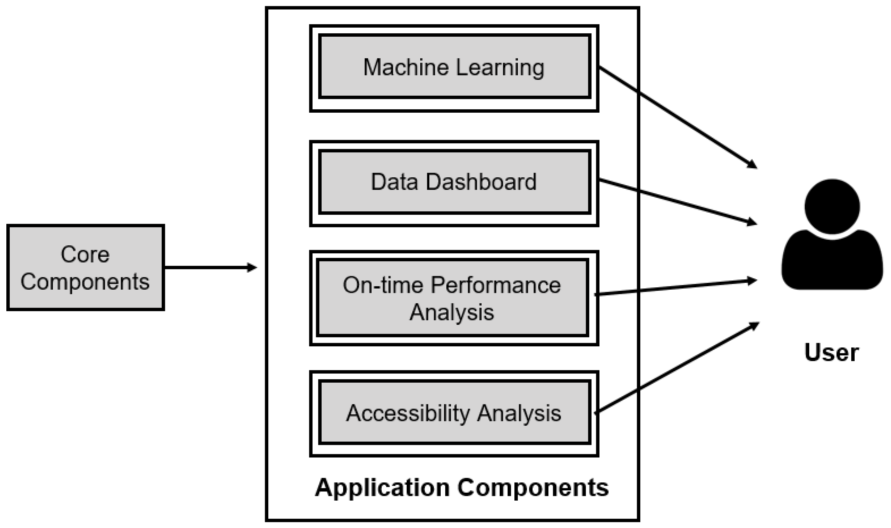


Figure 3: Diagram of Application Components

The Application Components that we have developed are divided into four subsections. Many other types of applications can be supported within this same architecture. The four Application Components being discussed are:

- Machine Learning Pipeline
- Data Dashboard
- On-time Performance Analysis
- Accessibility Analysis

The Application Components are utilized by end users of system (e.g., via a web interface). Each of these existing Application Components target a different type of user. The Machine Learning Pipeline may be of interest to researchers (and subsequently transit agencies) who are interested improving arrival and departure predictions. The Data Dashboard is intended for system administrators so they can add new feeds and understand how system resources are being managed for existing feeds being archived. The On-time Performance Analysis component may be of interest of transit agencies who would like to assess and improve operation of their vehicles on various routes. Accessibility Analysis may be of interest of Metropolitan Planning Organizations or transit planners who and considering adding new transit routes or modifying existing transit routes. We discuss in more detail about On-Time Performance Analysis in section 2 and Accessibility Analysis in section 3.

The data archiving system described in section 1 is the foundation for the analyses in section 2 and 3. For the remainder of this report, we will refer to this system as Transit Ecosystem Archiving (TEA).

2.3 SYSTEM IMPLEMENTATION

The system architecture presented in the previous section has been implemented by the research team and has recorded eight months of GTFS and GTFS-RT feeds for seven different transit agencies. During the implementation process, it became apparent that modern concepts in hosted applications such as containerization, microservices, and parallelization, discussed in detail below, were critical to the success of the system.

We present the implementation for archiving data from a single transit agency first, followed by a description of how that same design is leveraged in parallel for multiple agencies.

Single Transit Agency

Each Core component is implemented as a service of a system. Some services such as the GTFS-RT Recorder and Data API run continuously while others such as the GTFS-RT Archive and GTFS Recorder are executed on a schedule. This approach allows a simpler modular implementation in software source code that is easier to understand, allows for multiple teams to work on the same project, and allows institutions to selectively deploy the services they are interested in.

Containerization, which has become a popular implementation technique for hosted services in the past few years, is used to simplify the installation and management of individual services. There are three major container-based features which support the system presented in this paper: sandboxing, network abstraction and file system abstraction.

- Sandboxing – This feature allows a program to run in a virtualized environment. Within a sandbox an application developer can include an instruction file of how the sandbox is meant to be setup. This includes the base operating system, any required packages for the program to run and a copy of the program itself. As a result, after installing a Container manager daemon such as Docker [12] each service will automatically download and install all of their dependencies needed to execute on an environment independent from the host operating system. This allows system deployers to utilize a different container manager system or hosted service (e.g., Amazon, Microsoft, Google) without needing to change implementation details of the system itself.
- Network abstraction – This feature allows the control of networking for each individual program. Virtual Networks can be easily configured to allow secure communications

between the relevant services internally on a machine. Additionally, certain services can be made available from outside the virtual network. Within the scope of the project, this feature allows for the Data API to become one of the entry points of the network while protecting the database behind the virtual network.

- File system abstraction – As part of a sandboxed environment, the program within the container is not granted access to the physical disk and instead is seen as the only program executing on the machine. This unfortunately implies that the host operating system does not have access to the virtual filesystems for each of the containers. A solution is provided by the container manager where directories from the physical machine can be symbolically linked to a location a virtual filesystem. This feature is utilized for implementing support of multiple transit agencies.

Container-based features are able to solve issues such as installation requirements, security issues, and dependency conflicts, but still leaves the task of deploying several containers to the entity deploying the system. This can be simplified further with the utilization of Container Orchestrators such as Docker Compose [13], which is used in the author's implementation, and Kubernetes [14]. Orchestrators are used to deploy a group of containers that are related to each other, their network, variables and the directories each container has access to within the physical disk. Container Orchestrators reduce the effort required to deploy a system that is able to record, archive and access transit data to a simple execute command for the Container Orchestrator that is provided a single configuration file.

Multiple Transit Agencies

With the logistics of running a single-agency system addressed, the solution to support multiple transit agencies becomes simplified with a container-based solution. Due to the isolated nature of the virtualized environments, running multiple instances of the same component does not cause issues during execution. File system abstraction is leveraged by allowing multiple instances of a component to access the same location on their virtual disk that are connected to different locations within the physical disk. When this feature is augmented by the use of environment variables, containers of the same software can be used to perform the same task over various datasets. Depending on the amount of transit agencies the systems is monitoring, scalability can become a cause of concern, but is relieved with the use of orchestrators meant for distributed systems such as Docker Swarm and Kubernetes. Orchestrators such as these would automatically deploy containers to different nodes of a distributed system automatically while maintaining network and file system abstraction from the programs.

2.4 ESTIMATED COST FOR MAINTAINING THE DATA ARCHIVAL SYSTEM

We have estimated the cost for maintaining the new data archival system by measuring deployment metrics per month and extrapolate them to a year. The system presented in the previous section have been deployed to record GTFS and GTFS-RT feeds for seven different transit agencies. Table 1 outlines the disk usage growth on a per-month basis.

Table 1: Deployment metrics on archived data for seven transit agencies

Agency	Location	Polling (seconds)	Size (per month)	Max Vehicles	Per-stop predictions
Hillsborough Area Regional Transit (HART)	Tampa, FL, USA	30	200MB	130	No
Massachusetts Bay Transportation Authority (MBTA)	Boston, MA, USA	60	8GB	102	Yes
Halifax Transit (METROTRANSIT)	Halifax, NS, CA	15	5.50GB	245	Yes
Pinellas Suncoast Transit Authority (PSTA)	Pinellas County, FL, USA	35	1GB	172	Yes
Central Puget Sound Regional Transit Authority (SOUNDTRANSIT)	Seattle, WA, USA	5	2GB	264	Yes for some TripUpdates feeds
VIA Metropolitan Transit (VIA)	San Antonio, TX, USA	30	8GB	368	Yes
USF Bull Runner Shuttle	Main USF campus area, Tampa, FL, USA	30	6MB	12	N/A (No TripUpdates feed)

We then used these measurements to estimate the cost of hosting the system on cloud servers. Four cloud services were considered: Digital Ocean, Amazon Elastic Compute Cloud (EC2), and Microsoft Azure. From our observations, the core components of the system are neither processor-intensive nor memory-intensive, but require significant storage space. Based on this experience and the information provided in Table 1, we assume that a virtual machine with 4 Gigabytes memory and 2 virtual CPUs is a reasonable choice to run the system. In terms of storage, an initial disk space of 500 Gigabytes is required to store the data that has been archived since December 2017, and new data is expected to occupy additional 75 Gigabytes per month.

We looked for pricing information from Digital Ocean [15] and Amazon Elastic Compute Cloud (EC2) [16] for the aforementioned specifications. Since our system is designed on Docker Container platform, Amazon Elastic Container Service [17] is also a viable option. We also contacted Microsoft Azure for pricing. The results of pricing are listed in Table 2. With a discounted 3-year pricing package, Amazon was the most cost-effective at $3,504.00 total for 3 years and is the recommended provider.

Table 2: Estimated Cost on Cloud Platforms

Cloud Service	Memory (GB)	Number of Virtual CPUs	Total Cost Per 12 months	Total Cost per 36 months
Digital Ocean	4	2	$1533.00	$4,599.00
Amazon Elastic Compute Cloud	4	2	$1638.00	$3,504.00
Microsoft Azure	3.5	1	$1686.56	$5059.68

Since our system is designed based on container services, Amazon Elastic Container Service (ECS) is also a viable option. ECS is a logical grouping of Amazon Elastic Compute Cloud machines and uses Docker to instantiate containers on these EC2 hosts. We discovered that ECS is not particularly suitable for the design of our system because of the large number of containers in the system. There are two problems with using ECS to host our system. First, we have to configure the specification of each of the EC2 hosts according to each container. For each agency, we need a GTFS-Realtime archiving container and an API container. Micromanaging the specification for each of these containers is very inefficient, especially when we are trying to make the system scalable to multiple agencies. Second, we have attempted to assign some low-specification virtual machines to these containers and estimate the cost, and the combined cost of all mini-EC2 machines significantly exceed the cost of one big EC2 machine presented in Table 2. The estimated cost for ECS is presented in Table 3.

Table 3: Estimated Cost of Amazon Elastic Container Service

Instances	Quantity	Memory	vCPUs	EBS Storage	Memory Cost per sec	vCPU Cost per sec	Run Time per month	Total cost per month
Web server container	1	0.5 MB	1	0	$0.00000353	$0.00001406	2592000	$41.02
Database container	1	3.75 MB	1	1000 GB	$0.00000353	$0.00001406	2592000	$170.76
GTFS-RT archive container	6	0.5 MB	1	0	$0.00000353	$0.00001406	518400	$8.20
Retrieval container	6	0.5 MB	1	0	$0.00000353	$0.00001406	518400	$8.20
Total cost per month								$228.18
Total cost per year								$2,738.17
Total cost for 3 years								$8,214.51

3. PRODUCING RETROSPECTIVE GTFS PACKAGE FROM ARCHIVED GTFS-REALTIME DATA

3.1 THE NEED FOR PRODUCING A GTFS PACKAGE FROM REAL-TIME INFORMATION

GTFS data has allowed researchers to investigate interesting problems, such as measuring disparities in service provision [18, 19], temporal variability [20], the role of relative travel times and costs in mode choice [21, 22], the degree of accessibility offered by competing transit development plans [23], and many others. However, such research using GTFS is subject to a serious criticism: it is based entirely on scheduled data, which are expectations about service, rather than real observations of the actual performance of the transit network. It is common knowledge that transit does not run precisely as scheduled, and that it often differs substantially from the schedule. Occasionally there are major unscheduled disruptions due to severe congestion, vehicle breakdowns, or signal malfunctions. These disruptions are a fact of life for most transit users and well acknowledged by transit agencies themselves. One way that agencies have acknowledged service delays and disruptions is to issue live updates to their transit schedules via Service Alert entities in a GTFS-Realtime feed. The real-time information that contains vehicle location reports based on the Automatic Vehicle Location (AVL) systems and arrival predictions have been made available via Vehicle Locations and Trip Updates entities in a GTFS-Realtime feed. However, how such real-time information differs from scheduled data and how the difference affects the analyses that are based on schedule data alone remains largely unknown. Wessel et al. made a significant contribution to investigating this problem by describing a method for retroactively creating a GTFS package by using real-time vehicle location data available from the NextBus Application Programming Interface (API) [5], which is a way to access real-time data made available by the company NextBus for cities where their automatic vehicle tracking technology is deployed. Considering the popularity of GTFS-realtime (67 transit agencies publishing GTFS-realtime feeds according to TransitFeeds.com [9] as of 2019) and the fact that GTFS-realtime is a de facto standard implemented by many automatic vehicle location system vendors, the transit community is motivated in having a method to generate a GTFS package from GTFS-realtime data. Such a tool would help researchers capture the reality of a transit system and help transit agencies improve schedule reliability.

3.2 ASSUMPTIONS FOR RETRO-GTFS FROM GTFS-REALTIME

We assume that the following inputs are available:

- A set of archived GTFS-Realtime Vehicle Positions entities. In addition, we further assume that timestamp and trip_id are available for each vehicle in the entities. As of version 2.0, the vehicle timestamp and trip_id are optional fields in the GTFS-Realtime Specification whereas the feed header's timestamp is required.
- A valid GTFS package of the same transportation system. A valid GTFS package would contain at least six files: agency.txt, stops.txt, routes.txt, trips.txt, stop_times.txt, and either calendar.txt or calendar_dates.txt. For the Retro-GTFS algorithm described below, only stops.txt, routes.txt, trips.txt, and stop_times.txt are necessary. shapes.txt can be leveraged if available but is not required.

Given the inputs above, the goal is to create a GTFS package whose stop_times.txt has arrival time and departure time that represent when the vehicle actually arrived and departed at the stop, instead of when it was scheduled to arrive and depart.

3.3 A NEW ALGORITHM FOR RETRO-GTFS BASED ON WESSEL ET AL.'S ALGORITHM

The key difference between our study and Wessel et al.'s is that a data set of archived GTFS-Realtime always has its corresponding GTFS dataset where we can obtain more information about trips to make the output more definitive and comparable to the original GTFS package. The information that we can use from a corresponding GTFS dataset (by matching the trip_id in the GTFS-realtime feed) that the NextBus API lacks are:

- Stop sequence for each trip - This allows the output Retro-GTFS package to have the exact same stop sequence as the original GTFS package for all trips. In Wessel et al., stop sequence needed to be inferred based on how close stops are to the path of a trip and therefore may differ from the scheduled data.
- Expected vehicle travel path for a trip from shapes.txt file - This information can significantly improve the process of estimating arrival times at a stop. In Wessel et al., a vehicle's path on a trip is inferred by Open Source Routing Machine (OSRM) [24], which assumes the vehicle takes the shortest path given data from OpenStreetMap. This inferred path can be incorrect if the route pattern does not follow the shortest path for any reason, or if the street network data in OpenStreetMap is incomplete or incorrect. GTFS shapes.txt is a canonical representation of the planned vehicle travel path provided by the agency themselves. Even though a shapes.txt file is optional, this information can save processing time and increase accuracy if it is available.

- A list of trips and their scheduled operating days available from trips.txt and calendar.txt. This information allows us to aggregate multiple daily Retro-GTFS files into one package.

Figure 4 summarizes the modified Retro-GTFS algorithm created for this project to leverage GTFS and GTFS-realtime data. The white blocks indicate parts that remain the same as in Wessel et al, and the blue blocks indicate parts that have been modified.

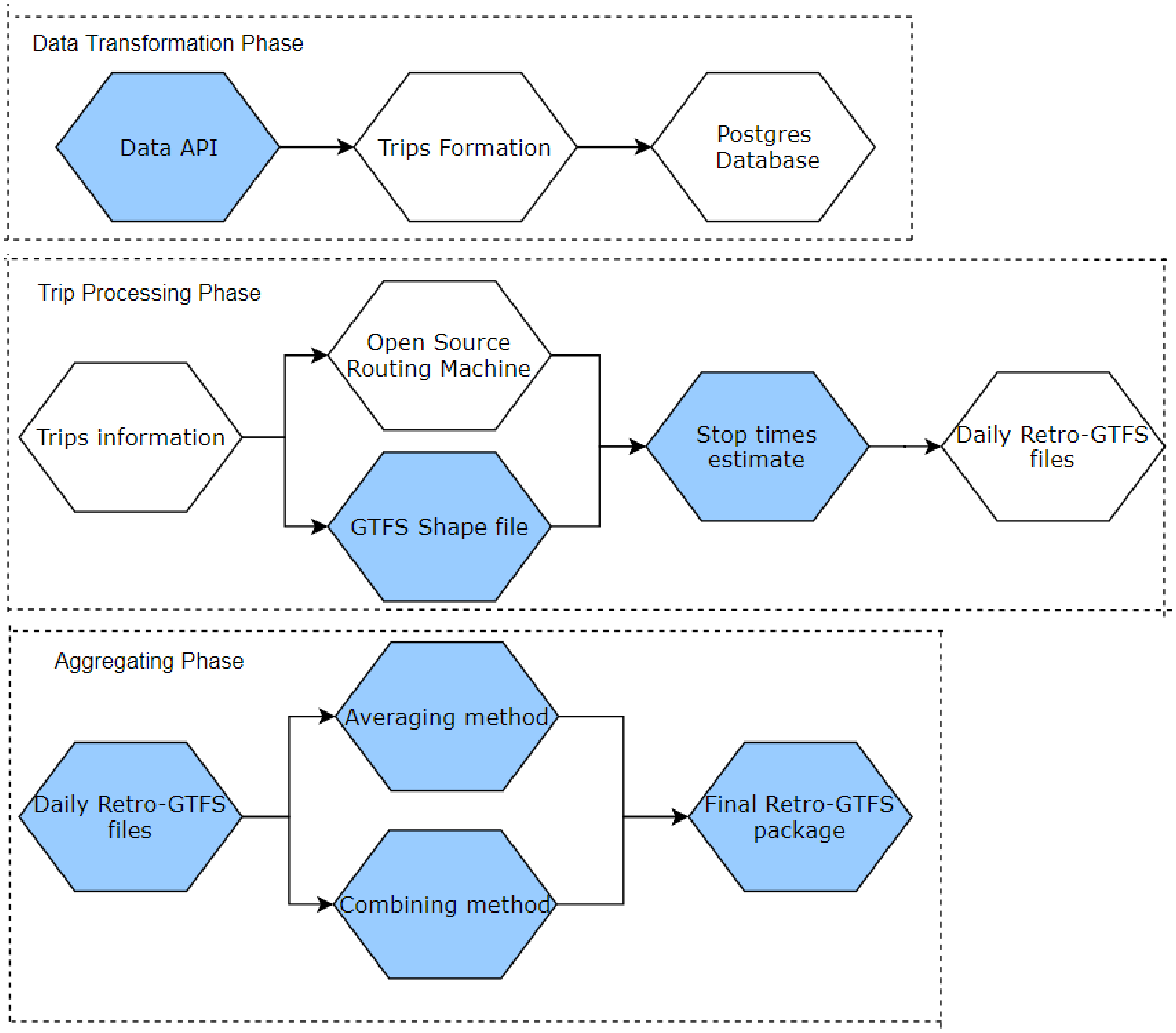


Figure 4: Retro-GTFS Process

The algorithm consists of three phases:

- Data Transformation Phase - Since the data from the Data API are raw GTFS-Realtime feeds at multiple points of time in the past, it is inefficient to use this data for processes such as OSRM and estimating stop times. In this phase, we transform the raw data into trip objects that consist of vehicle locations and corresponding time stamps. The Data API from the Core Components section of the data archival system described earlier in this report was used to demonstrate this part of the algorithm, but the archived GTFS-realtime data could come from any source.

- Trip Processing Phase - The main purpose of this phase is to use the information available from trip objects and the GTFS package to estimate arrival time to stops on trips. Trips are processed daily and a Retro-GTFS package is created for each day that consist of the same routes.txt, trips.txt, stops.txt, and calendar.txt that are the identical as in the original GTFS package, and a new stop_times.txt that is inferred from the real-time data. The reason for splitting this process daily is that in GTFS data convention, trips are not repeated within the same day. In other words, with scheduled transit service one trip_id cannot refer to more than one trip on the same day, but trip_ids can be repeated over multiple days.
- Aggregating Phase - The main purpose of this phase is to combine Retro-GTFS packages from multiple days into one single Retro-GTFS packages that can be used in on-time performance assessment or accessibility analysis.

These phases are sequential, meaning that the output of Data Transformation Phase is the input for Trip Processing Phase, and the output of Trip Processing Phase is the input for Aggregating Phase. Details for each phase are discussed below.

3.3.1 Data Transformation Phase

In this phase, our program retrieves all GTFS-Realtime Vehicle Positions entities in a day from the archiving system's Data API. We assume that each vehicle reported in a feed entity that is operating a trip is assigned with the corresponding trip_id from GTFS, and that any vehicle without trip_id is not operating a revenue trip. From the set of GTFS-Realtime data, we then create trip objects that consist of vehicle locations and corresponding time stamp. A GTFS-Realtime Vehicle Positions feed does not report when a vehicle finishes a trip. Therefore, we define a trip ending by checking one of the following two criteria:

- The vehicle's trip_id has changed
- The vehicle has not been seen for more than 30 minutes

All the trip objects are then stored in a Geographic Information System (GIS)-enabled Postgres database.

This phase is identical as Wessel et al.'s description except that our data source is archived GTFS-Realtime feeds whereas Wessel et al. created the trip objects in real time as data is being requested from the NextBus API.

3.3.2 Trip Processing Phase

In this phase, we attempt to estimate arrival and departure times for stops along a trip. To improve the spatial resolution of the data throughout the duration of the recorded trips, and to catch any GPS measurement errors, we need to match the sampled locations of the trip to a plausible route on the network. There are two ways to achieve this:

- The preferred and easier way is to use the shapes.txt file if it is provided in the original GTFS package. The shapes.txt file describes the physical path that a vehicle takes as a set of ordered latitude and longitude coordinates, and tracing the coordinates in order would provide the vehicle's path. This is a part where our algorithm differs from Wessel et al.'s.
- If shapes.txt file is not provided in the original GTFS package, we match our trips to a combination of roads and streetcar tracks with detailed data from OpenStreetMap using the Open-Source Routing Machine (OSRM) [24], an open-source routing software. OSRM accepts a series of points and timestamps matches the route to the provided network. For each returned match, OSRM also estimates the probability that the returned route is the correct one based on a number of factors using a naive Bayes classifier.

Next, we need to identify which stops the vehicle visited during the trip. Wessel et al. did this by looking at how close stops are to the matched path from OSRM. However, this approach is prone to both false positive (stops determined to be visited but were not actually visited) and false negative (stops determined to not be visited but were actually visited). By using the stop sequence information from stop_times.txt in the GTFS package, we can determine exactly the visited stops and their order. In observations by the research team, this change in approach reduces calculation time and may increase the accuracy in stop time estimates.

After the stop sequence is determined, stop times are estimated by projection and interpolation. GPS points are first projected on to the path, these projection points are then used to interpolate time at stops. For example, in Figure 5, the arrival time at stop S1 is calculated as:

$$Time\ at\ P1 + (Time\ at\ P2 - Time\ at\ P1)\frac{Distance(P1', S1)}{Distance(P1', P2')}$$

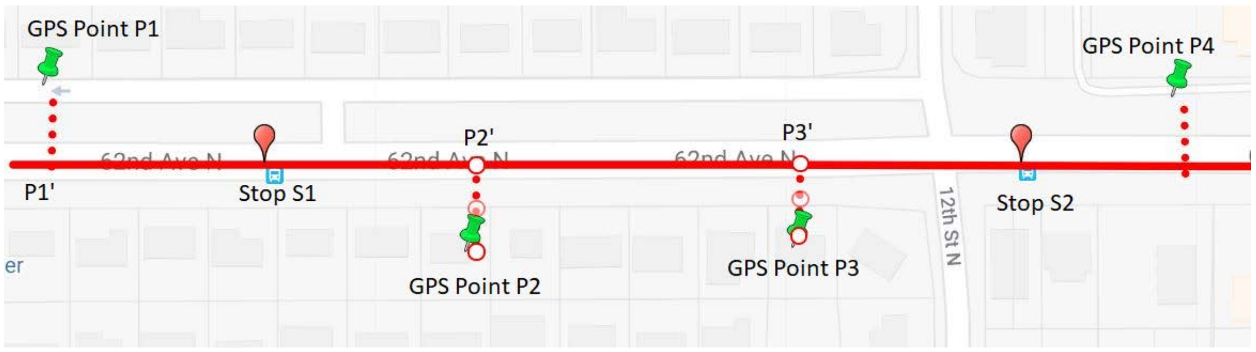


Figure 5: Example of Projection and Interpolation

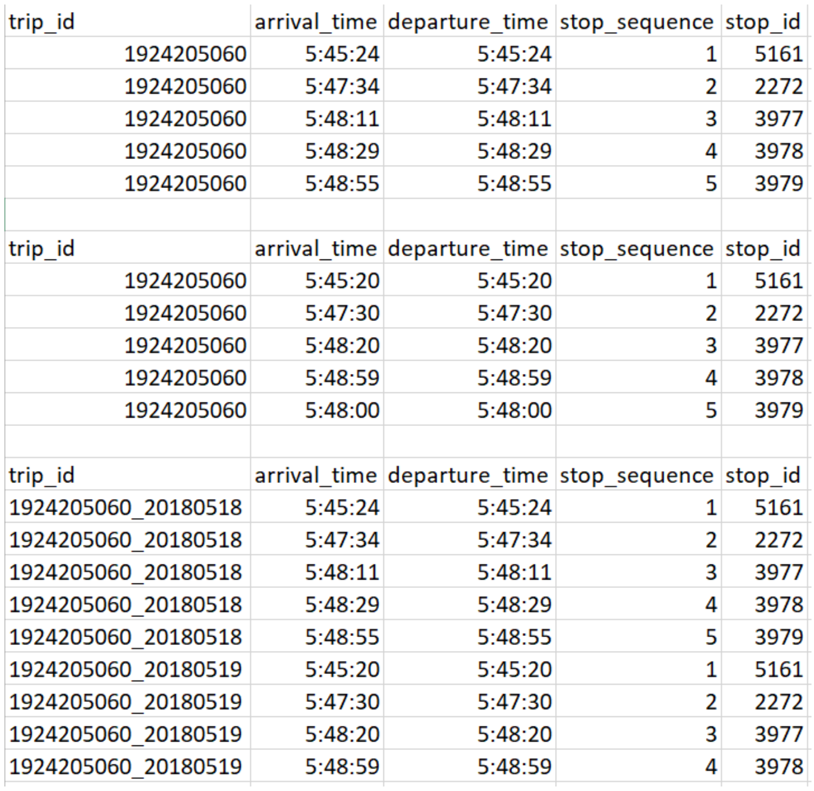
Green pins represent the vehicle GPS locations being projected to the red expected vehicle travel path, and red balloon markers represent the stop locations. This projection and interpolation step are done the same way as Wessel et al.

3.3.3 Aggregating Phase

The Aggregating Phase is new in our approach and is now possible because the same trip on different days have the exact same stop sequence with each other and with the original GTFS package. This is accomplished with the modification that we made in Section 2.3.2 of matching the trip_ids between the GTFS and GTFS-realtime data to assign observed real-time vehicles to specific GTFS trips.

Since Retro-GTFS packages from all days have the same routes.txt, trips.txt, stops.txt, and calendar.txt, we only need to process stop_times.txt when generating a Retro-GTFS package that represents actual system performance over an extended time period. There are two ways to achieve this:

1. Aggregate real-time arrivals and departures - For each pair of trip_id and stop_sequence in stop_times.txt, find all the arrival times and departure times in the daily GTFS files, then produce a summary statistic on those values (e.g., take their averages). For example, trip_id 1916000 at stop_sequence 1 has arrival_time of "09:00:00" in day 1 and "09:01:00" in day 2, the arrival_time value in the aggregated stop_times.txt is "09:00:30". This process produces a smaller GTFS file, but outlier performance on specific days would be lost or smoothed over by the summary statistic.
2. Bundle all real-time arrivals and departures - Treat all trips as unique trips by modifying the trip_id from all days and using calendar_dates.txt instead of calendar.txt. This practice in GTFS omits calendar.txt, and specify each date of service in calendar_dates.txt to allow for considerable service variation and accommodates service without normal weekly schedules. For example, the first two tables in Figure 6 show the results of the same trip on 05/18/2018 and 05/19/2018, and how they are combined in the aggregated file presented in the third table.



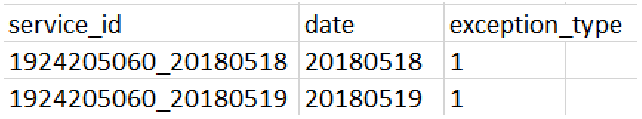
trip_id	arrival_time	departure_time	stop_sequence	stop_id
1924205060	5:45:24	5:45:24	1	5161
1924205060	5:47:34	5:47:34	2	2272
1924205060	5:48:11	5:48:11	3	3977
1924205060	5:48:29	5:48:29	4	3978
1924205060	5:48:55	5:48:55	5	3979

trip_id	arrival_time	departure_time	stop_sequence	stop_id
1924205060	5:45:20	5:45:20	1	5161
1924205060	5:47:30	5:47:30	2	2272
1924205060	5:48:20	5:48:20	3	3977
1924205060	5:48:59	5:48:59	4	3978
1924205060	5:48:00	5:48:00	5	3979

trip_id	arrival_time	departure_time	stop_sequence	stop_id
1924205060_20180518	5:45:24	5:45:24	1	5161
1924205060_20180518	5:47:34	5:47:34	2	2272
1924205060_20180518	5:48:11	5:48:11	3	3977
1924205060_20180518	5:48:29	5:48:29	4	3978
1924205060_20180518	5:48:55	5:48:55	5	3979
1924205060_20180519	5:45:20	5:45:20	1	5161
1924205060_20180519	5:47:30	5:47:30	2	2272
1924205060_20180519	5:48:20	5:48:20	3	3977
1924205060_20180519	5:48:59	5:48:59	4	3978

Figure 6: Example of keeping all trips unique

And the calendar_dates.txt would look as in Figure 7.



service_id	date	exception_type
1924205060_20180518	20180518	1
1924205060_20180519	20180519	1

Figure 7: Example of calendar_dates.txt

This process produces a very large GTFS file but does not lose any resolution – the exact observed arrival and departure time for every stop on every trip for every day is retained in the data.

4. APPLICATION OF RETRO-GTFS FOR ON-TIME PERFORMANCE

We have successfully applied the modified Retro-GTFS algorithm to archived GTFS-Realtime data for the Pinellas Suncoast Transit Authority (PSTA) transit agency in Pinellas County, Florida from 5/18/2018 to 6/16/2018. Table 4 presents a comparison between the scheduled arrival time in GTFS package and the averaged arrival time from daily Retro-GTFS files for one of PSTA's trips. The 'count' column shows the number of times the trip appeared between 5/18/2018 and 6/16/2018, which is also the number of daily arrival times that was averaged.

Table 4: Comparison between GTFS and Retro-GTFS

trip_id	arrival_time	stop_id	stop_sequence	count	Retro-GTFS's arrival_time	Arrival Difference (seconds)
1921429010	7:05:20	1781	1	5	7:05:00	-20.00
1921429010	7:05:50	1782	2	5	7:05:33	-17.00
1921429010	7:06:00	1783	3	5	7:05:50	-10.00
1921429010	7:06:18	1784	4	5	7:06:26	8.00
1921429010	7:06:51	1786	5	5	7:07:28	37.00
1921429010	7:07:15	1787	6	5	7:08:09	54.00
1921429010	7:07:34	1788	7	5	7:08:47	73.00
1921429010	7:07:41	1789	8	5	7:09:01	80.00
1921429010	7:07:52	9991	9	5	7:09:19	87.00
1921429010	7:08:00	9986	10	5	7:09:32	92.00
1921429010	7:08:17	9987	11	5	7:09:59	102.00
1921429010	7:08:45	1793	12	5	7:10:36	111.00
1921429010	7:09:03	1794	13	5	7:11:06	123.00
1921429010	7:09:20	1795	14	5	7:11:32	132.00
1921429010	7:10:18	1796	15	5	7:13:17	179.00
1921429010	7:12:06	1797	16	5	7:16:34	268.00
1921429010	7:12:21	1798	17	5	7:16:53	272.00
1921429010	7:12:44	1799	18	5	7:17:27	283.00
1921429010	7:13:22	1800	19	5	7:18:24	302.00
1921429010	7:14:07	1801	20	5	7:19:56	349.00
1921429010	7:14:36	1802	21	5	7:20:53	377.00
1921429010	7:15:02	9951	22	5	7:21:32	390.00
1921429010	7:15:39	1804	23	5	7:22:17	398.00
1921429010	7:16:11	9922	24	5	7:23:05	414.00
1921429010	7:16:32	1806	25	5	7:23:49	437.00
1921429010	7:16:46	1807	26	5	7:24:19	453.00
1921429010	7:17:09	1808	27	5	7:24:54	465.00
1921429010	7:18:11	2	28	5	7:27:13	542.00

1921429010	7:19:24	1240	29	5	7:28:41	557.00
1921429010	7:25:13	1809	30	5	7:30:00	287.00

As we can see from Table 4, toward the end of the trip, the vehicle is late on average by more than 9 minutes. This type of comparison study would be particularly useful for transit agencies who would like to improve their operation or revise scheduled data. In addition, any research that requires a GTFS package as an input would benefit from this program as it would provide an insight into how the research's result would look based on actual transit system performance, instead of schedule transit system performance. An example of such a research is presented in the next section.

The modified Retro-GTFS program is available as open-source software at https://github.com/CUTR-at-USF/retro-gtfs/pull/1.

5. EVALUATING PUBLIC TRANSPORTATION IN TERMS OF ACCESSIBILITY

5.1 CURRENT TOOLS FOR ACCESSIBILITY ANALYSIS

An accessibility analysis measures the ability of the public to access jobs and other resources by using the public transportation network. Conveyal Analysis is an open-source tool developed by the company Conveyal that can integrate GTFS transit data, OpenStreetMap's world map, and Census data to assess the transit system's accessibility with various scenarios [6]. Conveyal Analysis is capable of doing a single point analysis and a regional analysis. A single point analysis calculates the total number of a given amenity that can be reached from a given location within a given amount of time by a combination of the provided transit system and walking, biking, and driving. A single point analysis is displayed on an isochrone map, showing how fast a person could get on a map within a given travel time.

Since the Conveyal Analysis tool was originally designed to use a scheduled GTFS package as an input, it is subject to an underlying assumption that most transit analyses do: the transit system operate precisely as scheduled. This assumption, as we know, is usually not the case. However, we can overcome this assumption by using the technique discussed in Chapter 3. As described in Chapter 3, the Retro-GTFS method can create a GTFS package that is entirely based on the archived real-time transit data and has the exact same structure as a scheduled GTFS package. Therefore, by using the Retro-GTFS package in the accessibility analysis and any other analysis, we can conveniently avoid the assumption that the transit system operate precisely as scheduled, and the analysis' result would reflect how the system works in actuality. In addition, by comparing an analysis that uses the scheduled GTFS data and an analysis that uses Retro-GTFS data (archived real-time data), we can observe how significantly the assumption affects the accessibility results.

4.2 USING REAL-TIME DATA FOR ACCESSIBILITY ANALYSIS

In order to use Retro-GTFS data for Conveyal Analysis, a user can simply upload the Retro-GTFS package instead of the scheduled GTFS package to the program. We have modified the Conveyal Analysis tool so that it searches for the retro-GTFS package on disk instead of asking user to upload a GTFS package. This modified version is available at https://github.com/CUTR-at-USF/analysis-backend/pull/1. Future work could integrate the Conveyal Analysis program into the system so that it automatically uses the outputs of the data archival system and Retro-GTFS to perform analysis.

As mentioned in Chapter 4, we have successfully applied the modified Retro-GTFS algorithm to archived GTFS-Realtime data for the Pinellas Suncoast Transit Authority (PSTA) transit agency in Pinellas County, Florida from 5/18/2018 to 6/16/2018. We then used both the resulting Retro-GTFS real-time package and the original GTFS schedule package for the Conveyal Analysis program. Figure 8 shows results of accessibility analysis for Pinellas County Transit Authority (PSTA) transit agency comparing actual system performance (archived real-time data) vs. the scheduled system performance (scheduled GTFS data).

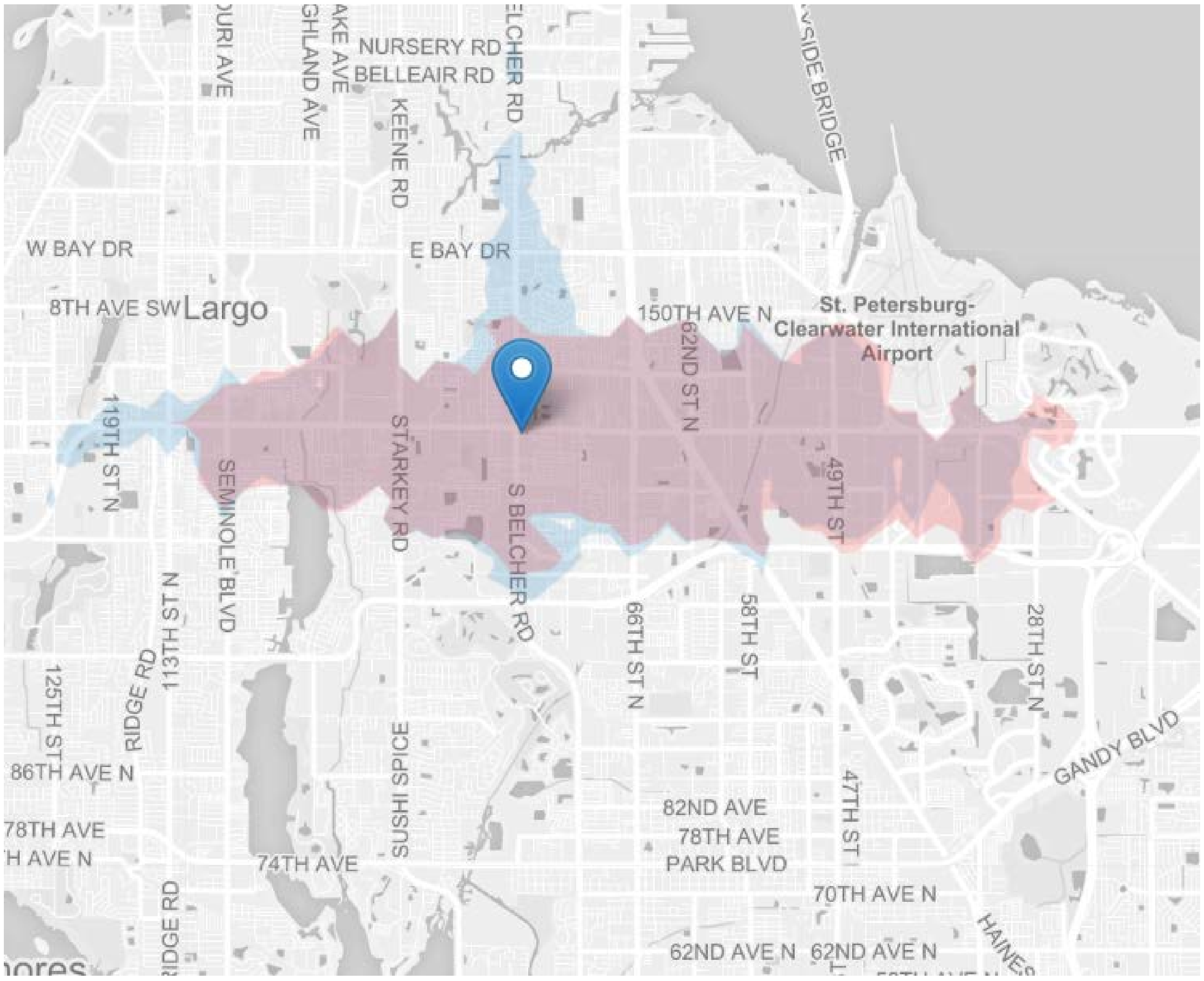


Figure 8: Conveyal Analysis results from scheduled GTFS and Retro-GTFS

The blue region represents region that is reachable within 45 minutes from the map marker when using the GTFS schedule data. The red region represents region that is actually reachable within 45 minutes from the blue mark when the actual system performance (observed real-time data) is used. There are significant sections of blue coverage so the north and west that appeared to be served within 45 minutes when only considering the schedule, but those areas are not in fact reachable in this amount of time based on actual system performance. Conversely, there appear to be areas to the east of the route where the vehicle runs ahead of schedule (where the red extends beyond the blue coverage) and riders can actually get further in the allotted 45 minutes than the schedule indicates. As a result, the red region, which is generated from Retro-GTFS (archived real-time information), should be used for decision-making instead of the blue region when considering accessibility in planning transit service.

The tools developed in this project will make this type of actual vs. schedule performance analysis much easier to perform and should help transit agencies better understand where service can be improved and adjusted to better serve riders.

6. CONCLUSION

We have introduced a novel system for archiving, retrieval, and analysis of real-time public transportation data. Implemented as a group of components, the framework can process data from multiple agencies concurrently while providing ample options for scalability. The system is made available as an open-source project to encourage active collaboration with other practitioners, and operations personnel. Furthermore, the system can support a centralized infrastructure where researchers can retrieve archived data and corroborate techniques across several datasets.

We have used the system presented in this paper to record eight months of GTFS and GTFS-RT feeds from seven different transit agencies, successfully illustrating the archival capabilities of the system. From the archived data, we have estimated the cost for hosting the system over an extended period of time (Chapter 2.4), showed a significant difference between the actual performance and scheduled data (i.e., on-time performance in Chapter 4), and compared the difference in accessibility analysis between when using scheduled data and when using archived real-time data (Chapter 4.2).

Any researcher can easily deploy this system to a local computer or to the cloud to record other transit agencies. Transit agencies may also use our system to archive their own GTFS-RT data, and to generate machine-learning predictions for arrival time. Metropolitan planning organizations may also be interested in using recorded real-time transit data from our system for their accessibility calculations instead of scheduled transit data. Recording all existing transit agencies globally is also possible if given sufficient data storage (e.g., cloud hosting). Due to the design of the architecture, changes to the system to allow new data sources or leveraging the system for other applications are relatively straightforward tasks, enabling the next generation of rapid analysis of actual transit system performance by planners, researchers, and analysts.

In addition, we have also established a method to generate a retrospective GTFS package from archived GTFS-realtime data, thus allowing analysis to leverage the large amount of existing GTFS-based tools on archived real-time data. This tool would be very useful for transit agencies, metropolitan planning organizations, and researchers who want to base their work on data for actual transit system performance, not scheduled transit system performance. Transit agencies can use this tool to assess their system's on-time performance. Researchers and metropolitan planning organizations can use this tool to perform analyses that are based on how the transit system actually operated.

The data archiving system and all the analysis tools presented in this report are available as open-source software and can be easily deployed for new systems. The public repositories URLs are as follows.

- Data archiving system: https://gitlab.com/cutr-at-usf/transi/getting-started
- Retro-GTFS using GTFS-Realtime: https://github.com/CUTR-at-USF/retro-gtfs/pull/1

- Modifications to Conveyal Analysis: https://github.com/CUTR-at-USF/analysis-backend/pull/1

Overall, the system created in this project could be the foundation for many useful applications.

Currently the data archival system is being hosted on USF lab computers. To be a long-term community resource, future work should move this system to be hosted on cloud resources (e.g., Amazon Web Services) and identify a long-term funding source for the hosting resources. Chapter 2.4 presents estimated costs for cloud hosting. Future work for this project can also focus on the machine learning pipeline to better understand what features are most useful for tasks such as generating predictions from archived data or improving the design of transit service, as well as the establishment of an on-line learning process by which the system can continuously improve based on new data.

References

1. Tufte, K.A., R.L. Bertini, and M. Harvey, *Evolution and Usage of the Portal Data Archive: 10-Year Retrospective.* Transportation Research Record: Journal of the Transportation Research Board, 2015(2527): p. 18-28.
2. Furth, P.G., et al., *Using archived AVL-APC data to improve transit performance and management.* 2006.
3. Bregman, S., *NJ TRANSIT to use real-time data to improve on-time performance*, in *The Transit Wire*. 2014.
4. Cevallos, F., *Transit Service Reliability: Analyzing Automatic Vehicle Location (AVL) Data for On-Time Performance and Identifying Conditions Leading to Service Degradation.* 2016.
5. Wessel, N., J. Allen, and S. Farber, *Constructing a routable retrospective transit timetable from a real-time vehicle location feed and GTFS.* Journal of Transport Geography, 2017. **62**: p. 92-97.
6. *Conveyal Analysis*. Jul. 19, 2018]; Available from: https://www.conveyal.com/analysis/.
7. Czebotar, j. *GTFS Data Exchange*. 2018 2018]; Available from: http://www.gtfs-data-exchange.com/.
8. Wong, J.C., *Use of the general transit feed specification (GTFS) in transit performance measurement*. 2013, Georgia Institute of Technology.
9. TransitFeeds. *About TransitFeeds*. 2018 2018]; Available from: https://transitfeeds.com/about.
10. TransitLand. *A Community-Edited Data Service Aggregating Transit Networks Across Metropolitan And Rural Areas Around The World*. 2018 2018]; Available from: https://transit.land/.
11. Google. *GTFS Realtime Overview*. Google Transit APIS 2018 2018]; Available from: https://developers.google.com/transit/gtfs-realtime/.
12. Merkel, D., *Docker: lightweight linux containers for consistent development and deployment.* Linux Journal, 2014. **2014**(239): p. 2.
13. Gerlach, W., et al. *Container orchestration for scientific workflows*. in *2015 IEEE International conference on cloud engineering*. 2015. IEEE.
14. Kubernetes. *What is Kubernetes?* Jul. 30, 2018]; Available from: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes.
15. DigitalOcean. Pricing on DigitalOcean - Cloud virtual machine & storage pricing 2018]; Available from: https://www.digitalocean.com/pricing/.
16. Services, A.W. *Simple Montly Calculator*. 2018]; Available from: https://calculator.s3.amazonaws.com/index.html.
17. Services, A.W. *Amazon Elastic Container Service*. 2018]; Available from: https://aws.amazon.com/ecs/.
18. Farber, S., B. Ritter, and L. Fu, *Space–time mismatch between transit service and observed travel patterns in the Wasatch Front, Utah: A social equity perspective.* Travel Behaviour and Society, 2016. **4**: p. 40-48.
19. Fransen, K., et al., *Identifying public transport gaps using time-dependent accessibility levels.* Journal of Transport Geography, 2015. **48**: p. 176-187.
20. Farber, S., M.Z. Morang, and M.J. Widener, *Temporal variability in transit-based accessibility to supermarkets.* Applied Geography, 2014. **53**: p. 149-159.

21. Owen, A. and D.M. Levinson, *Modeling the commute mode share of transit using continuous accessibility to jobs.* Transportation Research Part A: Policy and Practice, 2015. **74**: p. 110-122.
22. Salonen, M. and T. Toivonen, *Modelling travel time in urban networks: comparable measures for private car and public transport.* Journal of transport Geography, 2013. **31**: p. 143-153.
23. Farber, S. and M.G. Marino, *Transit accessibility, land development and socioeconomic priority: A typology of planned station catchment areas in the Greater Toronto and Hamilton Area.* 2017.
24. *Project OSRM.* 2019]; Available from: http://project-osrm.org.

CENTER FOR TRANSPORTATION EQUITY, DECISIONS & DOLLARS

The **Center for Transportation, Equity, Decisions and Dollars (CTEDD)** is a USDOT University Transportation Center, leading **transportation policy research** that aids in **decision making** and **improves economic development** through **more efficient, and cost-effective use of existing transportation systems,** and offers **better access to jobs and opportunities.** We are leading a larger consortium of universities focused on **providing outreach** and **research to policy makers**, through **innovative methods** and **educating future leaders of the transportation field.**

UNIVERSITY OF TEXAS ARLINGTON

CAL POLY SAN LUIS OBISPO

Georgia Tech

USF UNIVERSITY OF SOUTH FLORIDA

WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

C-TEDD@UTA.EDU CTEDD.UTA.EDU @UTACTEDD @C__TEDD @ CENTER FOR TRANSPORTATION, EQUITY, DECISIONS AND DOLLARS 817.272.5138