Overcoming Barriers for the Wide-scale Adoption of Standardized Real-time Transit Information

Sean J. Barbeau
University of South Florida

Follow this and additional works at: https://pdxscholar.library.pdx.edu/trec_reports

Part of the Transportation Commons, and the Urban Studies Commons

Let us know how access to this document benefits you.

Recommended Citation
Barbeau, Sean J. Overcoming Barriers for the Wide-scale Adoption of Standardized Real-time Transit Information. NITC-RR-1062. Portland, OR: Transportation Research and Education Center (TREC), 2018. https://doi.org/10.15760/trec.199

This Report is brought to you for free and open access. It has been accepted for inclusion in TREC Final Reports by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.
FINAL REPORT

Overcoming Barriers for the Wide-scale Adoption of Standardized Real-time Transit Information

NITC-RR-1062  April 2018

NITC is a U.S. Department of Transportation national university transportation center.
OVERCOMING BARRIERS FOR THE WIDE-SCALE ADOPTION OF STANDARDIZED REAL-TIME TRANSIT INFORMATION

Final Report

NITC-RR-1062

by

Dr. Sean J. Barbeau
Center for Urban Transportation Research
University of South Florida

for

National Institute for Transportation and Communities (NITC)
P.O. Box 751
Portland, OR 97207

April 2018
Real-time transit information has many benefits to transit riders and agencies, including shorter perceived and actual wait times, a lower learning curve for new riders, an increased feeling of safety, and increased ridership. In the last few years, a real-time complement to the General Transit Feed Specification (GTFS) format, GTFS-realtime, has emerged. GTFS-realtime has the potential to standardize real-time data feeds and lead to widespread adoption for transit agencies and multimodal apps. However, GTFS-realtime v1.0 has suffered from a lack of clear documentation and openly available validation tools, which significantly increases the time and effort necessary to create and maintain GTFS-realtime feeds. More importantly, bad data have been shown to have a negative effect on ridership, the rider’s opinion of the agency, and the rider’s satisfaction with multimodal apps.

This project focused on the community-driven creation of the GTFS-realtime v2.0 format, which establishes better guidance for transit agencies, application developers, and automatic vehicle location (AVL) system vendors on what fields are required or optional under various transit use cases. The research team also collaborated with the GTFS community to create GTFS Best Practices. In parallel to these standardization efforts, an open-source GTFS-realtime validation tool (https://github.com/CUTR-at-USF/gtfs-realtime-validator) was developed to allow these same parties to quickly identify and resolve problems in a feed. To demonstrate the utility of the GTFS-realtime Validator and capture the current state of real-time data quality in the industry, the Transit Feed Quality Calculator (https://github.com/CUTR-at-USF/transit-feed-quality-calculator) tool was created to automatically download and validate a large number of agency feeds. An evaluation of 78 transit agency GTFS-realtime feeds showed errors in 54 feeds and warnings in 58 feeds, indicating widespread problems with quality control across many agencies and AVL vendors.

Future work should focus on encouraging agencies to use GTFS-realtime v2.0 and the GTFS-realtime Validator, especially when specifying requirements in RFPs for new AVL systems. A hosted instance of the GTFS-realtime Validator would be useful for agencies that cannot run the tool themselves (e.g., due to internal IT policies preventing installation of applications). Official GTFS-realtime Best Practices voted upon by the GTFS community, similar to GTFS Best Practices, should also be created. Some gray areas remain in the GTFS-realtime specification that should be clarified via future proposals to the GTFS community, and new validator rules based on these clarifications could also be created. A data dashboard that shows the current quality of industry feeds may help agencies and vendors better understand how they relate to their peers in terms of data quality. GTFS may benefit from a more formal governance structure going forward, while being careful not to abandon key qualities of the grassroots approach to governance that has served the format well to date. Finally, the research into how the adoption of GTFS-realtime v2.0 and the GTFS-realtime validator impact data quality over time could be examined, as well as the possible institutional barriers that prevent some agencies from acknowledging and resolving errors in the data.
ACKNOWLEDGEMENTS

This project was funded by the National Institute for Transportation and Communities (NITC) under grant number NITC-RR-1062. The PI would like to acknowledge the Google Summer of Code program, which funded Nipuna Gunathilake’s work on a very early version of the open-source GTFS-realtime Validator software tool. Thanks to Mohan Gandhi Achchakkagari and Surya Vamshi Kandukoori for their software development work on the GTFS-realtime Validator during this project, and Surya Vamshi Kandukoori for his contributions to the open-source Transit Feed Quality Calculator tool as well as his efforts in collecting, analyzing, and visualizing errors and warnings found in feeds.

DISCLAIMER

The contents of this report reflect the views of the authors, who are solely responsible for the facts and the accuracy of the material and information presented herein. This document is disseminated under the sponsorship of the U.S. Department of Transportation University Transportation Centers Program and Florida Department of Transportation in the interest of information exchange. The U.S. Government and Florida Department of Transportation assume no liability for the contents or use thereof. The contents do not necessarily reflect the official views of the U.S. Government and Florida Department of Transportation. This report does not constitute a standard, specification, or regulation.

RECOMMENDED CITATION

Barbeau, Sean J. Overcoming Barriers for the Wide-scale Adoption of Standardized Real-time Transit Information. NITC-RR-1062. Portland, OR: Transportation Research and Education Center (TREC), 2017.
# TABLE OF CONTENTS

EXECUTIVE SUMMARY .......................................................................................................... 1  
1.0  INTRODUCTION ............................................................................................................. 6  
   1.1 GTFS AND GTFS-REALTIME FORMATS ................................................................. 7  
2.0  GTFS-REALTIME V2.0 .............................................................................................. 11  
   2.1 CHALLENGES WITH GTFS-REALTIME V1.0 ......................................................... 11  
   2.2 DEFINING TRANSIT-SPECIFIC FIELD REQUIREMENTS ....................................... 16  
3.0  IMPROVING GTFS DATA UNIFORMITY .................................................................... 19  
   3.1 CHALLENGES OF GTFS DATA UNIFORMITY ....................................................... 19  
   3.1.1 Feed management ............................................................................................... 19  
   3.1.2 Data content ...................................................................................................... 20  
   3.2 GTFS BEST PRACTICES ......................................................................................... 20  
   3.3 ADDITIONAL GTFS AND GTFS-REALTIME IMPROVEMENTS ........................... 22  
4.0  GTFS-REALTIME VALIDATION TOOL ................................................................... 23  
   4.1 GTFS-REALTIME VALIDATOR ............................................................................... 23  
   4.2 VALIDATION RULES .............................................................................................. 29  
   4.3 GTFS-REALTIME BATCH PROCESSOR .................................................................. 32  
5.0  ANALYSIS OF GTFS-REALTIME FEEDS ............................................................... 33  
6.0  CONCLUSIONS AND FUTURE WORK .................................................................... 37  
   6.1 FUTURE WORK ....................................................................................................... 37  
7.0  REFERENCES ............................................................................................................. 40
LIST OF TABLES

Table 1.1 - An example of data from the GTFS format stop_times.txt file................................. 8
Table 1.2 - An example of data from a GTFS-realtime Trip Update message......................... 9
Table 2.1 - A fully compliant GTFS-realtime v1.0 Vehicle Position........................................ 11
Table 2.2 - A missing stop_sequence value is problematic for routes with loops................... 12
Table 2.3 - An excerpt of the gtfs-realtime.proto file............................................................. 14
Table 4.1 - Errors currently implemented in the GTFS-realtime Validator............................ 29
Table 4.2 - Warnings currently implemented in the GTFS-realtime Validator......................... 30
Table 4.3 - Example output from the Batch Processor validation tool.................................... 32
LIST OF FIGURES

Figure 1 - Typical real-time information flow from a transit agency to a mobile app.................................10
Figure 2 - A route that includes a loop that visits stop_id A more than once..................................................12
Figure 3 - An example of the space savings of the binary Protocol Buffer format.........................................13
Figure 4 - The Protocol Buffer compiler autogenerates code to exchange binary GTFS-realtime messages....15
Figure 5 - Example of GTFS-realtime v2.0 showing new “Required” and “Cardinality” fields.......................17
Figure 6 - The stop_sequence field as defined in GTFS-realtime v2.0..........................................................18
Figure 7 - The new GTFS Best Practices, available at http://gtfs.org/best-practices/.................................21
Figure 8 - GTFS-realtime Validator tool source code on GitHub.................................................................23
Figure 9 - The first screen of the GTFS-realtime Validator..........................................................................24
Figure 10 - The monitoring screen, with a summary of all errors and a log of each feed iteration...............25
Figure 11 - Error and warning IDs (e.g., E002) shown next to every record in the Summary and Log sections26
Figure 12 - Two rules from the validator rule documentation on GitHub......................................................27
Figure 13 - The validator screen showing the ID of the last iteration and the time when it was validated ....27
Figure 14 - The Iteration Details screen shows the GTFS-realtime message (left) and occurrences of each error or warning (right).................................................................................................................28
Figure 15 - All errors and warnings for a feed message can be downloaded in CSV format........................29
Figure 16 - Industry-wide GTFS-realtime Feed Validation Results...............................................................34
Figure 17 - Most Frequent Errors and Warnings in GTFS-realtime Feeds..................................................34
Figure 18 - Distribution of Error Frequencies in GTFS-realtime Feeds.......................................................35
EXECUTIVE SUMMARY

Real-time transit information has been shown to have many benefits to transit riders, including shorter perceived wait time [1], shorter actual wait time [1], a lowered learning curve for new riders [2], and increased feeling of safety (e.g., at night) [3, 4]. Transit agencies that have deployed real-time information have also benefitted from increased ridership [5, 6], as well as a better perception of the agency and its transit service, even if its service hasn’t actually changed [7].

Availability of transit schedule, stop, and route information to transit riders via mobile apps has historically been driven by agencies sharing this data in the General Transit Feed Specification (GTFS) format [8], which has become the dominant format for open schedule data in the transit industry and shared by over 1,500 agencies worldwide [9].

In the last few years, a real-time counterpart to GTFS, GTFS-realtime [10], has begun to emerge, with agencies sharing their real-time data in this format. Previously, real-time transit information had only been shared in proprietary formats specific to each vendor or agency. GTFS-realtime offers the opportunity for application developers to create a mobile app that can function across a large number of cities and agencies, and for practitioners and researchers to be able to easily study and compare actual system performance across different transit systems using the same tools, without the overhead of manually transforming data into a consistent format. Having real-time transit data available in a common format is a key pillar for real-time multimodal information systems.

Data quality is of equal importance to data availability. In fact, accuracy of real-time information is a key concern of transit riders. A survey of riders of a mobile transit app showed that 84% rely solely on real-time information instead of using the schedule [4]. Errors in predictions create a negative perception of the mobile app providing the information as well as the transit agency. For example, 74% of surveyed Puget Sound transit riders considered a difference between actual and estimated arrival times greater than four minutes as an “error.” In addition, 9% of surveyed riders said that they took the bus less often due to errors they experienced [4]. Prediction errors can also lead to reduced system performance if transit agency operations personnel are making decisions based on this data.

This project focused on the collaborative creation of the GTFS-realtime v2.0 format with the GTFS community, which establishes better guidance for transit agencies, application developers, and automatic vehicle location system vendors on what fields are required or optional under various transit use cases.
GTFS-realtime uses Protocol Buffers, a binary format that compresses data, to exchange information between app developers and transit agencies. In GTFS-realtime v1.0, transit data field requirements were copied from the Protocol Buffer configuration file. Protocol Buffer configurations typically define most fields as “optional” for better forward compatibility (i.e., so software can be changed without breaking communication with older versions). However, this documentation error resulted in labeling many transit data fields as “Optional” when they should have been required under certain transit use cases. AVL vendors used this documentation when creating feeds, resulting in some data being erroneously omitted. GTFS-realtime v2.0 defines new properties for each field, including a new “Required” field that can have the values of Required, Conditionally Required, or Optional depending on the transit use case. Fields that are Conditionally Required have information in the Description property that defines when they are optional and when they are required. An example of a Conditionally Required field is that stop_sequence is now required within a trip_update if that trip has a loop that visits a stop more than once.

GTFS differs from other data standards in that it is largely a grassroots effort driven by transit agencies that produce the data as well as application developers who consume it in their applications – there is no official balloted standards organization (e.g., APTA, IEEE, ISO) that controls the format. GTFS has a community-driven process to adopt changes to the format, where anyone can propose a change (e.g., the changes for GTFS-realtime v2.0 created by the research team) that is then voted on by others in the community. However, there is a high bar for adoption of changes – a change must be implemented by both an agency as well as an application developer before a vote can be opened. Additionally, for a change to be adopted the vote must have unanimous approval – any votes against the proposal will prevent it from being adopted.

While the above governance model has served GTFS well in keeping it focused on a core set of services surrounding customer-facing trip planning and real-time information systems, it has resulted in some useful ideas that are endorsed by the majority, but not all, of those using GTFS being left out of the GTFS documentation. The research team collaborated with some of the most active members in the GTFS community [11] to create the GTFS Best Practices (http://gtfs.org/best-practices/) that are based on these concepts. These best practices help address some of the major challenges in data fragmentation and provide guidance to transit agencies, vendors, and consultants helping to produce and consume GTFS data, as well as app developers. GTFS Best Practices differ from the GTFS reference specification in that Best Practices only require a majority vote for adoption, while changes to the GTFS reference specification require unanimous consent. Additionally, GTFS Best Practices include guidance on how a feed should be managed and updated, which is beyond the scope of the GTFS reference specification that only governs the data format.
In parallel to these standardization efforts, an open-source GTFS-realtime validation tool (https://github.com/CUTR-at-USF/gtfs-realtime-validator) was developed to allow transit agencies, AVL vendors, and application developers to quickly identify and resolve problems in a GTFS-realtime feed. The GTFS-realtime Validator developed under this project was designed as an open-source tool to be downloaded and run on any computer. It includes a web application interface where users can enter a GTFS and GTFS-realtime URL and instantly get feedback on detected errors in the feed, including links to view the GTFS-realtime message that contains the error as well as the precision location in the feed where the error is contained. Users can share reports with others simply by copying and sharing the web application URL, or by clicking on a button to download the results to a comma-separated value (CSV) format. The GTFS-realtime Validator rules are implemented as a separate module within the project, which allows other developers to integrate these rules into their own application. A Batch Processor tool is also included within the library, which can use downloaded GTFS and GTFS-realtime data as input and then generate the validation results as output in the JavaScript Object Notation (JSON) format. These features allow other applications to integrate GTFS-realtime validation easily into their own systems.

To demonstrate the utility of the GTFS-realtime Validator and capture the current state of real-time data quality in the industry, the Transit Feed Quality Calculator (https://github.com/CUTR-at-USF/transit-feed-quality-calculator) tool was created to automatically download and validate many agency feeds. This tool retrieves a list of known GTFS-realtime feed URLs from TransitFeeds.com as well as a comma-separated value (CSV) file and downloads GTFS and GTFS-realtime data from each of these URLs. It then runs the GTFS-realtime Validator Batch Processor on all these feeds to generate an Excel file and JSON output with a summary of errors and warnings found in all of the analyzed feeds. An evaluation of 78 publicly available transit agency GTFS-realtime feeds (feeds without access restrictions listed in the feed directory TransitFeeds.com) showed integrity errors (e.g., trip IDs not matching GTFS, arrival predictions for stops that are out of order), in 54 feeds and warnings in 58 feeds, indicating widespread problems with quality control across many agencies and automatic vehicle location (AVL) vendors.

Now that the GTFS community has accepted GTFS-realtime v2.0, future work should focus on encouraging transit agencies to adopt it, especially when they create Requests for Proposals (RFPs) for new AVL systems. Possible strategies to increase awareness include announcements by government and industry organizations such as the Federal Transit Administration (FTA) and the American Public Transportation Association (APTA), as well as state departments of transportation and statewide public transportation organizations (e.g., FPTA). The research team created a presentation for GTFS-realtime v2.0 for a TransitCenter Transit Data Workshop [12] as well as a paper and poster for the 2018 Transportation Research Board conference [13] and a short blog post [14] announcing GTFS-realtime v2.0. Future conference presentations and webinars can utilize these same materials. Given that the U.S. Department of Transportation has
already requested GTFS data from agencies as part of the National Transit Map initiative [15], transit agencies may benefit from similar guidance regarding GTFS-realtime emerging as a de facto standard.

The research team has also identified additional issues that require clarification in the GTFS and GTFS-realtime specification [16]. Additional new rules (identified with the “new rule” label [17]) could also be added to the validator, including a few that first require proposals to clarify a portion of the GTFS-realtime documentation. Several transit agencies voiced the desire to have a hosted instance of the GTFS-realtime Validator tool instead of running it themselves. Future work should focus on establishing a long-term server and support where the GTFS-realtime Validator tool can be hosted [18].

Discussions with some transit agencies also uncovered a lack of knowledge for what is currently possible with GTFS-realtime. For example, some agencies were not aware that trips can be canceled using the current GTFS-realtime specification. Future work could focus on creating better documentation in layman’s terms that identifies these common use cases and clearly outlines what is and is not possible given the current specification. Future work could also focus on creating official GTFS-realtime Best Practices documentation via the GTFS community process, similar to how the GTFS Best Practices were established. For example, many of the warnings created for the GTFS-realtime validator are likely to be considered best and/or standard practices by the majority of the community and could be officially adopted via a vote.

The Transit Feed Quality Calculator tool can be improved – future work could focus on compiling additional feeds not currently documented in TransitFeeds.com, as well as utilizing a CSV file to store some URLs for feeds that require API keys [19]. Several agencies and application developers have expressed interest in a real-time “Data Dashboard” for GTFS-realtime feeds that would show the number of errors and warnings for all feeds for all agencies, as identified by the GTFS-realtime Validator. This dashboard may help an agency better understand how it is currently performing in real-time data quality, and how they compare to their peers. This same infrastructure can also collect and analyze data to drive evidence-based standard practices.

While the GTFS community has benefitted from the grassroots approach to governance in the past, given the substantial number of stakeholders for both GTFS and GTFS-realtime data the community may benefit from a more formal, organized structure going forward. The GTFS Best Practices effort showcases what can be accomplished relatively quickly by a smaller subset of organizations focused on accomplishing a specific goal, and a similar strategy will likely be required for creating GTFS-realtime Best Practices. This emerging coordination effort should be careful not to lose sight of the benefits of the grassroots approach that has made GTFS a widely adopted format (e.g., requiring producers and consumers for every new proposal to discourage speculative additions to the format). A more formal organization facilitating this coordination
could also host useful resources and tools for producers and consumers of the data (e.g., the
hosted GTFS-realtime Validator, the “Data Dashboard”).

Finally, there is research to be performed in the area of transit agency management and
supporting information technology related to institutional barriers of acknowledging and
resolving data problems. In the research team’s experience, even if a data error is reported to an
agency it is not always fixed. Additionally, some transit agencies fix data problems after they
are reported much more quickly than others. Identifying the successful practices that triage and
resolve data issues quickly may help other agencies that struggle with this process.
1.0 INTRODUCTION

Real-time transit information has been shown to have many benefits to transit riders, including shorter perceived wait time [1], shorter actual wait time [1], a lowered learning curve for new riders [2], and increased feeling of safety (e.g., at night) [3, 4]. Transit agencies that have deployed real-time information have also benefitted from increased ridership [5, 6], as well as a better perception of the agency and its transit service, even if its service hasn’t actually changed [7].

Availability of transit schedule, stop, and route information to transit riders via mobile apps has historically been driven by agencies sharing this data in the GTFS format [8], which has become the dominant format for open schedule data in the transit industry and shared by over 1,500 agencies worldwide [9]. In the last few years, a real-time counterpart to GTFS, GTFS-realtime [10], has begun to emerge, with agencies sharing their real-time data in this format. Previously, real-time transit information had only been shared in proprietary formats specific to each vendor or agency. GTFS-realtime offers the opportunity for application developers to create a mobile app that can function across a large number of cities and agencies, and for practitioners and researchers to be able to easily study and compare actual system performance across different transit systems using the same tools, without the overhead of manually transforming data into a consistent format. Having real-time transit data available in a common format is a key pillar for real-time multimodal information systems.

However, of equal importance to data availability is data quality. In fact, accuracy of real-time information is a key concern of transit riders. A survey of riders of a mobile transit app showed that 84% rely solely on real-time information instead of using the schedule [4]. Errors in predictions create a negative perception of the mobile app providing the information as well as the transit agency. For example, 74% of surveyed Puget Sound transit riders considered a difference between actual and estimated arrival times greater than four minutes as an “error.” In addition, 9% of surveyed riders said that they took the bus less often due to errors they experienced [4]. Prediction errors can also lead to reduced system performance if transit agency operations personnel are making decisions based on this data.

The following sections of this chapter present an introduction to the GTFS and GTFS-realtime formats. Subsequent chapters discuss how GTFS-realtime v2.0 and a new GTFS-realtime Validation tool can help agencies improve the quality of their real-time data.
1.1 GTFS AND GTFS-REALTIME FORMATS

GTFS forms the foundation for a GTFS-realtime feed – a GTFS-realtime feed cannot provide practical real-time prediction information without having a companion GTFS feed that defines the schedule. GTFS data is implemented as a set of comma-delimited text files added to a single zip file [8].

A subset of the full GTFS specification is required for a GTFS-realtime feed – the following are key for understanding real-time information:

- **stops.txt** – All bus stops included in a feed, with each record including a stop_id (identifier internal to agency), stop_code (rider-facing stop identifier), stop location, location_type (a single stop or station with multiple stops), etc. For some agencies, stop_id and stop_code may be the same.

- **routes.txt** – All routes defined for an agency, including a route_id and short and long name.

- **calendar.txt** and **calendar_dates.txt** – Includes service days and times, each identified via a service_id, that the agency provides service.

- **trip.txt** – All trips defined for an agency, including to which route_id each trip belongs. A route may have multiple trip patterns, depending on the day and/or time. The day/time that each trip is operational is defined by a service_id that relates to calendar.txt and/or calendar_dates.txt.

- **stop_times.txt** – The core schedule file that defines, for each trip_id, the ordered list of bus stops that will be visited, along with a scheduled arrival and departure time, and whether or not each stop is a timepoint (optional).
A stop_times.txt file will look like the following:

<table>
<thead>
<tr>
<th>trip_id</th>
<th>arrival_time</th>
<th>departure_time</th>
<th>stop_id</th>
<th>stop_sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2777</td>
<td>5:52:00</td>
<td>5:52:00</td>
<td>4301</td>
<td>1</td>
</tr>
<tr>
<td>2777</td>
<td>5:52:34</td>
<td>5:52:34</td>
<td>3471</td>
<td>2</td>
</tr>
<tr>
<td>2777</td>
<td>5:53:46</td>
<td>5:53:46</td>
<td>4456</td>
<td>3</td>
</tr>
<tr>
<td>2777</td>
<td>5:54:27</td>
<td>5:54:27</td>
<td>592</td>
<td>4</td>
</tr>
</tbody>
</table>

The GTFS-realtime specification can be broken down into three types of elements:

- **Trip Updates** – Real-time predictions for when vehicles arrive and depart. Predictions (stop_time_updates) are represented as an update to the time that the vehicle was scheduled to arrive or depart (defined in GTFS stop_times.txt), either as a relative “delay” or “time.” stop_time_updates are identified using a trip ID from GTFS trips.txt.

- **Vehicle Positions** – Real-time vehicle location, trip assignment (defined using the trip ID from GTFS trips.txt), and occupancy information.

- **Service Alerts** – Descriptions of events that affect transit service, along with the transit stops/routes that the event impacts. For example, “Route 5 is on detour due to flooding.”
A GTFS-realtime Trip Update for trip_id 2777 that predicts a bus running 60 seconds late for stop_id 4456 (stop_sequence 3), running on time for stop_id 592 (stop_sequence 4), and 60 seconds early for stop_id 593 (stop_sequence 5), would look like the following:

```cpp
Table 1.2 - An example of data from a GTFS-realtime Trip Update message

trip_update {
  trip {
    trip_id: "2777"
  }
  stop_time_update {
    stop_sequence: 3
    arrival {
      delay: 60 // Schedule deviation of 60 seconds (running late)
    }
    stop_id: "4456"
  }
  stop_time_update {
    stop_sequence: 4
    arrival {
      delay: 0  // Schedule deviation of 0 seconds (on time)
    }
    stop_id: "592"
  }
  stop_time_update {
    stop_sequence: 5
    arrival {
      delay: -60 // Schedule deviation of -60 seconds (running early)
    }
    stop_id: "593"
  }
}
```

The architecture of a real-time transit information system can be divided up into two components [20] as shown in Figure 1:

1. The Producer - The system generating the GTFS-realtime feed (typically the automatic vehicle location (AVL) system), labeled as the “Transit Agency Server” in Figure 1.

2. The Consumer – The system reading the GTFS-realtime feed (typically a server and mobile app displaying the information to a transit rider), labeled as the “App Developer Server” in Figure 1.
While GTFS datasets are typically updated 3-4 times per year (e.g., when new schedules are published), a GTFS-realtime Trip Updates and Vehicle Positions feed can be updated as often as every few seconds and are typically driven by an AVL system.

GTFS-realtime datasets are formatted in the Protocol Buffer format [21], which is a very efficient binary representation of the information in the feed. As a result, the actual GTFS-realtime messages produced and consumed by applications require special software to convert them to human-readable plain text.
2.0 GTFS-REALTIME V2.0

GTFS-realtime v2.0 accomplishes a major step forward in terms of better defining what information should be contained in a GTFS-realtime message. The following section discusses the limitations in GTFS-realtime v1.0 that drove the development of v2.0.

2.1 CHALLENGES WITH GTFS-REALTIME V1.0

Having a de facto standard for real-time information is beneficial to those who use the data – it lets app developers focus on creating new features instead of developing convertor software for each feed type and identifying and removing discrepancies in the data.

However, as more transit agencies and app developers started using GTFS-realtime, they noticed something peculiar – almost all of the GTFS-realtime fields were optional. To be exact, of the 63 GTFS-realtime data fields, only seven were required – about 11% of all fields [10].

The overwhelming number of optional fields makes it very simple for AVL system implementers to roll out a GTFS-realtime feed that is officially compliant with GTFS-realtime v1.0 – they can leave most of the values blank. However, it creates challenges when app developers start consuming that information, as some critical information may be missing. Transit riders are upset when an app gives them bad data. Information errors reflect poorly on the app developer and transit agency, and cause problems with the AVL vendor due to unmet data quality expectations.

As an example, Table 2.1 shows a fully compliant GTFS-realtime v1.0 feed for a vehicle position:

Table 2.1 - A fully compliant GTFS-realtime v1.0 Vehicle Position

```
header {
  gtfs_realtime_version: "1.0"
}
entity {
  id: "d131dd02"
  vehicle {
    position {
      latitude: 28.04265
      longitude: -82.45945
    }
  }
}
```
There is critical information missing:

- When was this position calculated?
- What route or trip is this vehicle currently serving?
- How do we describe the vehicle to a transit rider? Is d131dd02 a valid bus number?

A second example is shown in Table 2.2. When providing arrival predictions in GTFS-realtime v1.0, the stop_sequence field is optional:

```
Table 2.2 - A missing stop_sequence value is problematic for routes with loops
trip {
    trip_id: "277725"
}
stop_time_update {
    arrival {
        delay: 900 // 15 minutes
    }
    stop_id: "A"
}
```

This message that contains stop_id, but not stop_sequence, works fine for most routes. However, a missing stop_sequence value creates problems when you have a route with a loop that visits a stop more than once, such as that shown in Figure 2.

![Diagram of a route with a loop that visits stop_id A more than once](image)

**Figure 2 - A route that includes a loop that visits stop_id A more than once**
Without stop_sequence, the delay value of 15 minutes could potentially pertain to either the first time the vehicle arrives at Stop A or the second. This ambiguity creates a problem when the information is shown to riders. Riders waiting to board at Stop B will be upset if the app tells them they have time to grab an extra coffee (because it interprets the 15-minute delay being in the first half of the loop) and then they miss the bus when the vehicle arrives on time because the delay was actually in the second half of the loop. The stop_sequence field is required in this case to interpret the prediction correctly.

The labeling of optional fields in GTFS-realtime v1.0 documentation was driven by the use of Protocol Buffers [21] for compressing real-time data messages. Protocol Buffers are an extremely compact way to represent information in a binary format. Instead of the feed data being formatted as Unicode text characters as shown earlier in this report, each of which takes at least 1 byte (8 bits), it can be compressed into a smaller representation of 0s and 1s (i.e., bits).

The space savings of Protocol Buffers adds up quickly, especially considering that GTFS-realtime messages are exchanged every few seconds. For example, a single response from Massachusetts Bay Transportation Authority (MBTA)’s GTFS-realtime Trip Updates feed in the binary Protocol Buffer format is under 1 MB, while the plain text version is more than six times bigger at just over 5.5 MB.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>TripUpdates.pb</td>
<td>PB File</td>
<td>891 KB</td>
</tr>
<tr>
<td>TripUpdates.pb.txt</td>
<td>Text Document</td>
<td>5,683 KB</td>
</tr>
</tbody>
</table>

**Figure 3 - An example of the space savings of the binary Protocol Buffer format**

While binary formats are extremely space efficient, it can be time consuming to create software that processes them. Protocol Buffers solve this problem by generating this code automatically. First, the developer creates a .proto file that describes the data elements to exchange (in the case of GTFS-realtime, the official gtfs-realtime.proto, shown in Table 2.3).
Table 2.3 - An excerpt of the gtfs-realtime.proto file

```proto2
option java_package = "com.google.transit.realtime";
package transit_realtime;

message FeedMessage {
  // Metadata about this feed and feed message.
  required FeedHeader header = 1;
  // Contents of the feed.
  repeated FeedEntity entity = 2;
  ...
}

message FeedHeader {
  required string gtfs_realtime_version = 1;
  enum Incrementality {
    FULL_DATASET = 0;
    DIFFERENTIAL = 1;
  }
  ...
}

message FeedEntity {
  ...
  optional TripUpdate trip_update = 3;
  optional VehiclePosition vehicle = 4;
  optional Alert alert = 5;
  ...
}
```

Then, developers can input this .proto file into the open-source Protocol Buffer tools [21] to autogenerate the code that compresses the information to a binary format (used in the Transit Agency AVL server) and the code to extract it from a binary format (used in the App Developer’s server), as shown in Figure 4.
To make this process even more convenient, Google has already performed these steps and created a readily usable gtfs-realtime-bindings library [22] that supports easily exchanging GTFS-realtime messages in the programming languages Java, .NET, JavaScript / Node.js, PHP, Python, Ruby, and Golang.

The implementation of GTFS-realtime in the Protocol Buffer format was the reason behind the extensive number of fields marked “Optional” in GTFS-realtime v1.0. The GTFS-realtime spec includes a *Cardinality* field that was copied from the gtfs-realtime.proto file and does not have anything to do with public transportation. Protocol Buffer Cardinality simply defines whether or not software parsing the binary message expects a field to exist – it has no direct mapping to GTFS or transit-specific logic (repeated, in case you’re wondering, is Protocol Buffer-lingo for a list of optional elements). This becomes a problem in GTFS-realtime because many software engineers choose not to label any Protocol Buffer fields as Required because of forward compatibility issues with Protocol Buffer implementations (see the Protocol Buffer documentation “Required is Forever” [23] and a GTFS-realtime Google Group discussion [24] for details). As a result, nearly all fields in GTFS-realtime v1.0 are shown as “Optional,” even if that field is necessary for a transit app to show proper real-time information to a transit rider.
2.2 DEFINING TRANSPORT-SPECIFIC FIELD REQUIREMENTS

The research team worked with the GTFS-realtime community to create an updated version of the format that defines the semantic requirements and cardinality of real-time information to fix this confusion about which fields are optional and required. GTFS-realtime v2.0 [10] now defines which fields should be required based on domain-specific (i.e., transit) logic. The detailed proposal for defining semantic cardinality is available online [25]. These definitions are completely independent of the Protocol Buffer format and would still apply even if a different format was used for GTFS-realtime data.

In GTFS-realtime v2.0, each field now has a Required column that can contain the following values:

- Required: This field must be provided by a GTFS-realtime feed producer.

- Conditionally required: This field is required under certain conditions, which are outlined in the field Description. Outside of these conditions, the field is optional.

- Optional: This field is optional and is not required to be implemented by producers. However, if the data is available in the underlying automatic vehicle location systems (e.g., VehiclePosition timestamp) it is recommended that producers provide these optional fields when possible.

The Cardinality column now represents the number of elements that can be provided for a particular field – One or Many (e.g., a list of predictions applying to more than one stop within a trip).

Below in Figure 5 is a snapshot of what the GTFS-realtime FeedHeader looks like in GTFS-realtime v1.0 (top), and GTFS-realtime v2.0 (bottom):
### GTFS-realtime v1.0

**message FeedHeader**

Metadata about a feed, included in feed messages.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Required</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtfs_realtime_version</td>
<td>string</td>
<td>Required</td>
<td>One</td>
<td>Version of the feed specification. The current version is 1.0.</td>
</tr>
<tr>
<td>incrementality</td>
<td>Incrementality</td>
<td>Required</td>
<td>One</td>
<td>This timestamp identifies the moment when the content of this feed has been created (in server time). In POSIX time (i.e., number of seconds since January 1st 1970 00:00:00 UTC). To avoid time skew between systems producing and consuming real-time information it is strongly advised to derive timestamp from a time server. It is completely acceptable to use Stratum 3 or even lower strata servers since time differences up to a couple of seconds are tolerable.</td>
</tr>
<tr>
<td>timestamp</td>
<td>uint64</td>
<td>Required</td>
<td>One</td>
<td>Version of the feed specification. The current version is 2.0.</td>
</tr>
</tbody>
</table>

### GTFS-realtime v2.0

**message FeedHeader**

Metadata about a feed, included in feed messages.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Required</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtfs_realtime_version</td>
<td>string</td>
<td>Required</td>
<td>One</td>
<td>Version of the feed specification. The current version is 2.0.</td>
</tr>
<tr>
<td>incrementality</td>
<td>Incrementality</td>
<td>Required</td>
<td>One</td>
<td>This timestamp identifies the moment when the content of this feed has been created (in server time). In POSIX time (i.e., number of seconds since January 1st 1970 00:00:00 UTC). To avoid time skew between systems producing and consuming real-time information it is strongly advised to derive timestamp from a time server. It is completely acceptable to use Stratum 3 or even lower strata servers since time differences up to a couple of seconds are tolerable.</td>
</tr>
<tr>
<td>timestamp</td>
<td>uint64</td>
<td>Required</td>
<td>One</td>
<td>Version of the feed specification. The current version is 2.0.</td>
</tr>
</tbody>
</table>

Figure 5 - Example of GTFS-realtime v2.0 showing new “Required” and “Cardinality” fields
The new Required column can be seen in the v2.0 version of the documentation, which now makes critical fields like the header timestamp mandatory. This helps address the previously discussed problem of determining the age of a vehicle position.

The StopTimeUpdate message, which contains the information about arrival and departure predictions, is a good illustration of the new Conditionally required value:

![Message StopTimeUpdate](image)

In GTFS-realtime v2.0, stop_sequence is Conditionally required, and one of the required cases outlined in the Description is the loop, which addresses the problem with the ambiguous arrival prediction for the loop route presented earlier.

Figure 6 - The stop_sequence field as defined in GTFS-realtime v2.0
3.0 IMPROVING GTFS DATA UNIFORMITY

In parallel to the development of the GTFS-realtime v2.0, the research team contributed to an effort organized by the Rocky Mountain Institute to develop a set of GTFS Best Practices [26]. This effort attempted to address shared challenges that transit app developers encounter when working with GTFS data. Because GTFS-realtime requires GTFS data, these issues also impact the quality of a GTFS-realtime feed as well.

3.1 CHALLENGES OF GTFS DATA UNIFORMITY

Following are some examples of challenges that transit app developers have encountered in the past when trying to process GTFS data for a large number of sources.

3.1.1 Feed management

- Changing GTFS dataset URLs—In order to manage a large number of agencies in an app, app developers typically create software that automates the process of retrieving new GTFS datasets from agencies and integrating them into their application. If an agency puts their GTFS zip file at the location http://acmeagency.org/gtfs.zip, app developers will assume that new GTFS datasets will be uploaded to the same URL and will continuously poll that URL to see if the data changed. If an agency changes this URL for a new dataset (http://acmeagency.org/gtfs-March-2017.zip), the app will never know that this new data exists.

- Changing GTFS dataset IDs—A typical feature in transit apps are stop bookmarks so a user can easily retrieve information about their favorite stops. The app developer needs to have a way to track which stops the user has marked as their favorites. A simple way to do this is to save the stop_id of that stop to a database. Then, the next time the user wants to see arrival information for that stop, the saved stop_id is used to fetch real-time arrival information. This works well, until the next GTFS update when the stop_id for that stop changes. Now the user can’t retrieve arrival information for their favorite stop because that stop_id no longer exists.

- Gaps in GTFS data coverage—Agencies typically update their GTFS data on a quarterly basis. Therefore, agencies wait to do this until just before their previous data expires. For example, if changes go into effect April 1, an agency may publish their new GTFS dataset in the late hours of March 31. To make this data available to their users, app developers need to download the new dataset, validate it, add the new data to their
databases, and potentially push an update to all applications. This can definitely take more than a few hours to accomplish, which means that on April 1, when the user opens the app, no transit service will be visible. Agencies are able to avoid this issue by using the GTFS Merge Tool [27] to create a combined dataset for both the current and next schedule period, and by sharing/announcing this merged dataset to developers at least a week prior to the service change. This way, apps pick up the schedule change early and are ready to go when the change actually happens.

3.1.2 Data content

- **Case of text**—Some agencies PUBLISH ALL THEIR STOP NAMES, ROUTE NAMES, AND HEADSIGNS IN ALL CAPS! This makes text difficult to visually read in the app, and also makes it hard to distinguish between abbreviations and words.

- **Abbreviations**—When text is abbreviated, it’s meaning is not always clear. For example, is “Dr.” for “Doctor” or “Drive”? Sometimes we can tell the difference visually in context, but this is particularly challenging for text-to-speech engines (e.g., for accessibility, voice-driven personal assistants such as Amazon Alexa, Google Assistant, and Siri).

- **Loop routes**—If there is a route that continuously runs in a circle, should the first/last stop be included in the trip twice, or just once? The GTFS community found that the majority represent it with the same stop twice, once at the beginning of the trip and again at the end, but not everyone represents their data this way.

3.2 GTFS BEST PRACTICES

GTFS differs from other data standards in that it is largely a grassroots effort driven by transit agencies that produce the data as well as application developers who consume it in their applications – there is no official balloted standards organization (e.g., APTA, IEEE, ISO) that controls the format. GTFS has a community-driven process to adopt changes to the format, where anyone can propose a change (e.g., the changes for GTFS-realtime v2.0) that is then voted on by others in the community. However, there is a high bar for adoption of changes – a change must be implemented by both an agency as well as an application developer before a vote can be opened. Additionally, for a change to be adopted the vote must have unanimous approval – any votes against the proposal will prevent it from being adopted.

While the above governance model has served GTFS well in keeping it focused on a core set of services surrounding customer-facing trip planning and real-time information systems, it has resulted in some useful ideas that are endorsed by the majority, but not all, of those using GTFS being left out of the GTFS documentation. The research team collaborated with some of the
most active members in the GTFS community [11] to create the GTFS Best Practices (http://gtfs.org/best-practices/) that are based on these concepts.

GTFS Best Practices help address some of the major challenges in data fragmentation (including the above-mentioned items) and provide guidance to transit agencies and vendors/consultants helping to produce and consume GTFS data, as well as app developers that make the information available in their applications. GTFS Best Practices differ from the GTFS reference specification in that Best Practices only require a majority vote for adoption, while changes to the GTFS reference specification require unanimous consent. Additionally, GTFS Best Practices include guidance on how a feed should be managed and updated by a transit agency, which is beyond the scope of the GTFS reference specification that only governs the data format.

The GTFS Best Practices documentation is organized both by file from the GTFS dataset as well as by various use cases such as trip planners, human readability, arrival predictions, and timetables. This makes it simple for agencies and developers to understand what best practices should be followed for each file in the dataset, as well as the implications that various practices have on certain types of applications.

Figure 7 - The new GTFS Best Practices, available at http://gtfs.org/best-practices/
3.3 ADDITIONAL GTFS AND GTFS-REALTIME IMPROVEMENTS

In addition to the GTFS Best Practices, the research team also proposed several other improvements to the GTFS and GTFS-realtime formats and governance process that were voted on and accepted by the GTFS community, including the following:

- Clarify that a proposer's vote does not count towards total votes - https://github.com/google/transit/pull/50
- Specify that prior to a vote a GTFS producer/consumer should implement change - https://github.com/google/transit/pull/46
- Better cross-linking for a pull request and Google Groups post - https://github.com/google/transit/pull/35
- Add missing agency_email field - https://github.com/google/transit/pull/34
4.0 GTFS-REALTIME VALIDATION TOOL

While the GTFS format for schedule data has several open-source GTFS feed validators [28], no such open validation tool has existed for GTFS-realtime. The scale of the combined GTFS and GTFS-realtime datasets combined with the frequent refresh of real-time data makes manual inspection time and cost prohibitive. For example, in November 2017 Massachusetts Bay Transportation Authority (MBTA) in Boston [29] had a GTFS dataset that contained 71,260 trips and 1,809,833 stop time records. MBTA’s GTFS-realtime feed contains data for 489 vehicles with independent arrival or departure predictions for most stops on active trips that is refreshed around every five seconds. To effectively analyze this quantity and frequency of data, automated tools are required.

4.1 GTFS-REALTIME VALIDATOR

To address these problems, the research team created an open-source GTFS-realtime Validator software tool [30] that can monitor GTFS-realtime feeds (Trip Updates, Vehicle Positions, Service Alerts) and log any encountered problems. Instructions for running the GTFS-realtime Validator on any computer are found on the project README page on GitHub [30] (Figure 8).

Figure 8 - GTFS-realtime Validator tool source code on GitHub
To use the GTFS-realtime Validator, as shown in Figure 9, the user simply enters URLs for their GTFS and GTFS-realtime datasets, as well as how frequently the tool should fetch GTFS-realtime updates (the default is 10 seconds) and then click the “Start” button.

![GTFS-realtime Validator](image)

**Figure 9 - The first screen of the GTFS-realtime Validator**

After starting the monitoring session, the user is shown a “Monitoring Feeds” screen with three main sections: Overview, Summary, and Log (Figure 10).
Figure 10 - The monitoring screen, with a summary of all errors and a log of each feed iteration
The Overview screen provides basic information about how long the monitoring session has been running, along with the number of requests the Validator has made for GTFS-realtime feed updates as well as the number of unique responses that have been returned. If the Validator is polling the GTFS-realtime feed more quickly than it is updated, then duplicate responses will be returned to the Validator. In this case, the validator will detect the duplicate response and will not validate the same response more than once.

Upon each unique response from the GTFS-realtime server, each error and warning that is detected will be output to the Log section. The Log section serves as a historical record of all responses from the server in this session in reverse chronological order. The Summary section is shown above the Log. The Summary has one record for each error or warning that has been detected throughout this monitoring session, along with a count of the occurrences of this error or warning and a link to the last iteration that the error or warning was detected within. On the right side of the Summary section, each error or warning has an on/off switch for “Show in log.” The user can toggle this switch to filter our errors or warnings from the Log section that they do not want to see.

In both the Log and Summary sections, all error and warning IDs (e.g., E002) are hyperlinked (Figure 11).

![Figure 11 - Error and warning IDs (e.g., E002) shown next to every record in the Summary and Log sections](image)

When clicking on the error or warning ID, the user is directed to the “Implemented Rules” documentation on GitHub [31] where they can read more about that specific error or warning as well as follow reference links to sections within the GTFS-realtime specification related to each rule (Figure 12). For some rules, common mistakes that can cause a particular error are identified and possible solutions that can eliminate the error are provided.
E001 - Not in POSIX time

All times and timestamps must be in POSIX time (i.e., number of seconds since January 1st 1970 00:00:00 UTC).

Common mistakes - Accidentally using Java's `System.currentTimeMillis()`, which is the number of milliseconds since January 1st 1970 00:00:00 UTC.

Possible solution - Use `TimeUnit.MILLISECONDS.toSeconds(System.currentTimeMillis())` to convert from milliseconds to seconds.

References:
- `header.timestamp`
- `trip_update.timestamp`
- `vehicle_position.timestamp`
- `stop_time_update.arrival/departure.time`
- `alert.active_period.start` and `alert.active_period.end`

E002 - stop_time_updates not strictly sorted

stop_time_updates for a given `trip_id` must be strictly ordered by increasing `stop_sequence` - this also means that no `stop_sequence` should be repeated.

From Stop Time Updates description:
- Updates should be sorted by `stop_sequence` (or `stop_ids` in the order they occur in the trip).

From GTFS `stop_times.txt`:
- The values for `stop_sequence` must be non-negative integers, and they must increase along the trip.

This validation rule is implemented for both when `stop_sequence` is provided in the GTFS-rt feed, and when `stop_sequence` is omitted from the GTFS-rt feed.

Common mistakes - Assuming that the GTFS `stop_times.txt` file will be grouped by `trip_id` and sorted by `stop_sequence` - while sorting the data is a good practice, it's not strictly required by the spec.

Possible solution - Group the GTFS `stop_times.txt` records by `trip_id` and sort by `stop_sequence`. Also, make sure that no `stop_sequence` is repeated in GTFS `stop_times.txt`.

Figure 12 - Two rules from the validator rule documentation on GitHub

<table>
<thead>
<tr>
<th>Severity</th>
<th>Last iteration</th>
<th>Last time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>17</td>
<td>10:17:02 AM (1519139822)</td>
</tr>
<tr>
<td>ERROR</td>
<td>17</td>
<td>10:17:02 AM (1519139822)</td>
</tr>
</tbody>
</table>

Figure 13 - The validator screen showing the ID of the last iteration and the time when it was validated
In the Log and Summary sections, the Iteration ID for the GTFS-realtime message where the error or warning was detected is also hyperlinked (Figure 13). The user can click on this Iteration ID to see detailed information about this GTFS-realtime message, as shown in Figure 14.

The GTFS-realtime message that was retrieved from the feed for this iteration is shown on the left, and the occurrences of each error and warning are shown on the right. For example, the user can see that there is a total of seven occurrences of E002 “stop_time_updates not strictly sorted,” and on the right the specific trip_id that contained the error is shown along with the stop_sequences that appeared out-of-order. If the user wants to look at this trip_id in more detail, they can scroll the left side of the screen or use a keyboard shortcut such as Ctrl-F for Find to locate this particular trip_id within the GTFS-realtime message. If a user wants to share this information with others, they can simply copy the URL for this iteration and send it to another person (e.g., via email). As an alternative, they can click on the “Download all” button to download all the error and warning occurrences for this iteration into a comma-separated value
(CSV) file that can be opened in tools like Microsoft Excel (Figure 15). The user can also click on the download button for just a single error if they want to download only occurrences of that specific error.

Figure 15 - All errors and warnings for a feed message can be downloaded in CSV format

4.2 VALIDATION RULES

The GTFS-realtime Validator has a modular rule architecture that allows new errors and warnings to be easily added to the tool as the GTFS-rt specification continues to evolve and new problems are discovered in feeds. As of February 2018, the research team has implemented rules to detect 47 types of errors and nine types of warnings [31], many of which were encountered in the team’s experience analyzing real GTFS-realtime feeds. Tables of the currently implemented errors and warnings are shown in Table 4.1 and Table 4.2.

Table 4.1 - Errors currently implemented in the GTFS-realtime Validator

<table>
<thead>
<tr>
<th>Error ID</th>
<th>Error Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>E001</td>
<td>Not in POSIX time</td>
</tr>
<tr>
<td>E002</td>
<td>stop_time_updates not strictly sorted</td>
</tr>
<tr>
<td>E003</td>
<td>GTFS-rt trip_id does not exist in GTFS data</td>
</tr>
<tr>
<td>E004</td>
<td>GTFS-rt route_id does not exist in GTFS data</td>
</tr>
<tr>
<td>E006</td>
<td>Missing required trip field for frequency-based exact_times = 0</td>
</tr>
<tr>
<td>E009</td>
<td>GTFS-rt stop_sequence isn’t provided for trip that visits same stop_id more than once</td>
</tr>
<tr>
<td>E010</td>
<td>location_type not 0 in stops.txt</td>
</tr>
<tr>
<td>E011</td>
<td>GTFS-rt stop_id does not exist in GTFS data</td>
</tr>
<tr>
<td>E012</td>
<td>Header timestamp should be greater than or equal to all other timestamps</td>
</tr>
<tr>
<td>E013</td>
<td>Frequency type 0 trip_schedule_relationship should be UNSCHEDULED or empty</td>
</tr>
<tr>
<td>E015</td>
<td>All stop_ids referenced in GTFS-rt feeds must have the location_type = 0</td>
</tr>
<tr>
<td>Error ID</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>E016</td>
<td>trip_ids with schedule_relationship ADDED must not be in GTFS data</td>
</tr>
<tr>
<td>E017</td>
<td>GTFS-rt content changed but has the same header timestamp</td>
</tr>
<tr>
<td>E018</td>
<td>GTFS-rt header timestamp decreased between two sequential iterations</td>
</tr>
<tr>
<td>E019</td>
<td>GTFS-rt frequency type 1 trip start_time must be a multiple of GTFS headway_secs later than GTFS start_time</td>
</tr>
<tr>
<td>E020</td>
<td>Invalid start_time format</td>
</tr>
<tr>
<td>E021</td>
<td>Invalid start_date format</td>
</tr>
<tr>
<td>E022</td>
<td>Sequential stop_time_update times are not increasing</td>
</tr>
<tr>
<td>E023</td>
<td>trip_start_time does not match first GTFS arrival_time</td>
</tr>
<tr>
<td>E024</td>
<td>trip_direction_id does not match GTFS data</td>
</tr>
<tr>
<td>E025</td>
<td>stop_time_update departure time is before arrival time</td>
</tr>
<tr>
<td>E026</td>
<td>Invalid vehicle position</td>
</tr>
<tr>
<td>E027</td>
<td>Invalid vehicle bearing</td>
</tr>
<tr>
<td>E028</td>
<td>Vehicle position outside agency coverage area</td>
</tr>
<tr>
<td>E029</td>
<td>Vehicle position far from trip shape</td>
</tr>
<tr>
<td>E030</td>
<td>GTFS-rt alert trip_id does not belong to GTFS-rt alert route_id in GTFS trips.txt</td>
</tr>
<tr>
<td>E031</td>
<td>Alert informed_entity.route_id does not match informed_entity.trip.route_id</td>
</tr>
<tr>
<td>E032</td>
<td>Alert does not have an informed_entity</td>
</tr>
<tr>
<td>E033</td>
<td>Alert informed_entity does not have any specifiers</td>
</tr>
<tr>
<td>E034</td>
<td>GTFS-rt agency_id does not exist in GTFS data</td>
</tr>
<tr>
<td>E035</td>
<td>GTFS-rt trip.trip_id does not belong to GTFS-rt trip.route_id in GTFS trips.txt</td>
</tr>
<tr>
<td>E036</td>
<td>Sequential stop_time_updates have the same stop_sequence</td>
</tr>
<tr>
<td>E037</td>
<td>Sequential stop_time_updates have the same stop_id</td>
</tr>
<tr>
<td>E038</td>
<td>Invalid header.gtfs_realtime_version</td>
</tr>
<tr>
<td>E039</td>
<td>FULL_DATASET feeds should not include entity.is_deleted</td>
</tr>
<tr>
<td>E040</td>
<td>stop_time_update doesn't contain stop_id or stop_sequence</td>
</tr>
<tr>
<td>E041</td>
<td>trip doesn't have any stop_time_updates</td>
</tr>
<tr>
<td>E042</td>
<td>arrival or departure provided for NO_DATA stop_time_update</td>
</tr>
<tr>
<td>E043</td>
<td>stop_time_update doesn't have arrival or departure</td>
</tr>
<tr>
<td>E044</td>
<td>stop_time_update arrival/departure doesn't have delay or time</td>
</tr>
<tr>
<td>E045</td>
<td>GTFS-rt stop_time_update stop_sequence and stop_id do not match GTFS</td>
</tr>
<tr>
<td>E046</td>
<td>GTFS-rt stop_time_update without time doesn't have arrival/departure time in GTFS</td>
</tr>
<tr>
<td>E047</td>
<td>VehiclePosition and TripUpdate ID pairing mismatch</td>
</tr>
<tr>
<td>E048</td>
<td>header timestamp not populated (GTFS-rt v2.0 and higher)</td>
</tr>
<tr>
<td>E049</td>
<td>header incrementality not populated (GTFS-rt v2.0 and higher)</td>
</tr>
<tr>
<td>E050</td>
<td>timestamp is in the future</td>
</tr>
<tr>
<td>E051</td>
<td>GTFS-rt stop_sequence not found in GTFS data</td>
</tr>
</tbody>
</table>

Table 4.2 - Warnings currently implemented in the GTFS-realtime Validator

<table>
<thead>
<tr>
<th>Warning ID</th>
<th>Warning Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>W001</td>
<td>timestamps not populated</td>
</tr>
<tr>
<td>W002</td>
<td>vehicle_id not populated</td>
</tr>
<tr>
<td>W003</td>
<td>ID in one feed missing from the other</td>
</tr>
<tr>
<td>W004</td>
<td>vehicle speed is unrealistic</td>
</tr>
<tr>
<td>W005</td>
<td>Missing vehicle_id in trip_update for frequency-based exact_times = 0</td>
</tr>
<tr>
<td>W006</td>
<td>trip_update missing trip_id</td>
</tr>
<tr>
<td>W007</td>
<td>Refresh interval is more than 35 seconds</td>
</tr>
<tr>
<td>W008</td>
<td>Header timestamp is older than 65 seconds</td>
</tr>
<tr>
<td>W009</td>
<td>schedule_relationship not populated</td>
</tr>
</tbody>
</table>

An error is logged when data in the feed is incorrect and would result in a transit rider seeing bad or missing real-time information as a result. A warning is logged when a feed contains data that
would negatively affect some GTFS-rt consuming applications but either cannot be confirmed to be incorrect with 100% certainty or the GTFS-rt specification does not clearly indicate that the data or behavior is incorrect. For example, a vehicle speed value over 26 meters per second, or 60 miles per hour, is suspicious for most intercity buses but may be valid for high-speed rail. In another example of a warning, developers expect a GTFS-realtime feed to refresh its contents frequently (every 30-60 seconds), but the GTFS-rt specification doesn’t require a minimum update frequency.

Validation rules can be broken down into the following categories:

- **Header** – Checks if header fields (e.g., feed version) are populated correctly
- **Timestamps** – Checks integrity of feed timestamps (e.g., in POSIX format, age of feed, sequential arrival/departure times are in increasing order)
- **Stop Time Updates** – Checks the integrity of predictions provided for each trip (e.g., order by stop_sequence, missing field values, conflicts with GTFS stop_times.txt data)
- **Stops** – Checks that stop information provided in the feeds matches GTFS stops.txt (e.g., stop_id, location_type)
- **Trip Descriptors** – Checks integrity of trip properties (e.g., conflicts with GTFS data, missing data for certain use cases, trip start date formats)
- **Vehicle** – Checks integrity of vehicle properties (e.g., valid position/bearing formats, unrealistic speed values that may be unit conversion errors, proximity of real-time position to assigned GTFS trip)
- **Cross Feed** – If multiple feeds entity types exist (e.g., VehiclePositions and TripUpdates), checks if content in one set of entities matches the content in the other set of entities (e.g., that all trip_id and vehicle_id pairings are consistent)
- **Frequency Type Zero Trips** – Checks conditions specific to trips defined in frequencies.txt with exact_times = 0 (i.e., true frequency/headway-based service)
- **Frequency Type One Trips** – Checks conditions specific to trips defined in frequencies.txt with exact_times = 1 (i.e., scheduled service modeled using a specified headway interval)

A detailed description of all rules is documented on GitHub [31], and this GitHub page will continue to be updated as new rules are added to the validator. Detailed instructions for adding new rules are available on GitHub [32].
4.3 GTFS-REALTIME BATCH PROCESSOR

The rules implemented in the GTFS-realtime Validator can be used as a library in other software applications by referencing the gtfs-realtime-validation-lib project [33].

Within this library is the Batch Processor, which is a tool anyone can easily run from the command-line or from within another software application to process a GTFS and GTFS-realtime feed that has been downloaded to the computer. This application then outputs the validation results in JavaScript Object Notation (JSON) format (Table 4.3).

Table 4.3 - Example output from the Batch Processor validation tool

```json
[{
  "errorMessage": {
    "messageId": 0,
    "gtfsRtFeedIterationModel": null,
    "validationRule": {
      "errorId": "W001",
      "severity": "WARNING",
      "title": "timestamp not populated",
      "errorDescription": "Timestamps should be populated for all elements",
      "occurrenceSuffix": "does not have a timestamp"
    },
    "errorDetails": null
  },
  "occurrenceList": [
    {
      "occurrenceId": 0,
      "messageLogModel": null,
      "prefix": "trip_id 277716"
    },
    {
      "occurrenceId": 0,
      "messageLogModel": null,
      "prefix": "trip_id 277767"
    },
    {
      "occurrenceId": 0,
      "messageLogModel": null,
      "prefix": "trip_id 277768"
    }
  ]
},
...
```

Detailed documentation for using the Batch Processor is available in the README of the gtfs-realtime-validation-lib project [33].

The following chapter showcases an example application that uses the Batch Processor to validate a large number of feeds downloaded from various transit agencies.
5.0 ANALYSIS OF GTFS-REALTIME FEEDS

To demonstrate the industry need for the GTFS-realtime Validator, the research team developed another tool, the Transit Feed Quality Calculator [34], to automate the validation of a large number of live feeds from transit agencies.

This tool is easily executed from the command-line and:

- Retrieves the URLs for GTFS-realtime feeds and corresponding URLs for GTFS data from the TransitFeeds.com GetFeeds API (a centralized directory for GTFS and GTFS-realtime feed URLs)
- Downloads a snapshot of the GTFS-realtime and GTFS data from each agency's server into a subdirectory
- Runs the GTFS-realtime Validator on each of the subdirectories using the GTFS-realtime Validator Batch Processor library [33]
- Produces summary statistics and graphs for all validated feeds

While TransitFeeds.com shows a total of 130 GTFS-rt feeds that have been registered with the system [35], the team has so far been able to automate the validation of 78 feeds that are publicly available and do not have any access restrictions. As discussed later, future work will increase this number by supporting feeds that require application programming interface (API) keys or other types of authentication.

Out of the 78 feeds evaluated, 54 of the feeds contained errors and 58 of the feeds contained warnings (Figure 16).
Figure 16 - Industry-wide GTFS-realtime Feed Validation Results

Figure 17 - Most Frequent Errors and Warnings in GTFS-realtime Feeds
Figure 17 indicates the most common errors and warnings that appeared in feeds. “E011 – GTFS-rt stop_id does not exist in GTFS data” was the most common error, appearing in 16 feeds. E011 occurs when the GTFS schedule data has no record of a stop that the GTFS-rt data is showing a prediction for, indicating an incorrect stop_id either in the GTFS or GTFS-rt data. The second most common error was “E022 – Sequential stop_time_update times are not increasing” appearing in 15 feeds (which indicates that predicted times are wrong – the vehicle would be traveling backwards in time). “E045 - GTFS-rt stop_time_update stop_sequence and stop_id do not match GTFS” appeared in 13 feeds – this means that the GTFS-rt data shows a conflicting order of arrival for stops for a trip when compared to the GTFS data.

Figure 18 shows the distribution of the count of error types found in feeds. For example, the feed with the worst performance had seven different types of errors, while 23 feeds had only one error type. Even though the majority of feeds had two or fewer types of errors, as mentioned earlier, some errors can occur multiple times in the same feed iteration, as well as in multiple iterations of the feed. For example, in a feed that had eight different types of errors, there were 24 occurrences of “E022 – Sequential stop_time_update times are not increasing” in a single feed iteration. Each of these occurrences can have a significant impact on the transit rider experience, as the rider would be viewing incorrect or missing arrival predictions (in the case that the consuming app filters out data that is obviously wrong).

The above analysis is for a single iteration of each of the 78 evaluated feeds. It is highly likely that if the validator was executed over multiple iterations of the feed (e.g., several hours of time) additional errors and warnings would be found for each feed. Some errors may originate from mis-configured schedule or real-time data that could potentially be fixed once and only occur again if the schedule data changes (e.g., when the agency releases a quarterly GTFS data update).
Other errors may be harder to track down and could be related to bugs in the AVL vendor software or errors in vehicle operator logins that are less predictable in nature. As a result, it is recommended that agencies continuously monitor their real-time feeds so they are immediately aware of any problems that occur, especially when releasing a new GTFS dataset.
6.0 CONCLUSIONS AND FUTURE WORK

The quality of real-time transit information is important to transit riders and transit agencies. As mentioned earlier, 9% of surveyed riders said that they took the bus less often due to errors they experienced. Prediction errors can also lead to reduced system performance if the operations department is making decisions based on this data.

This project has helped fill a vital gap in the state of the art of real-time transit information. The research team proposed GTFS-realtime v2.0, which defines fields as Required, Conditionally Required, and Optional based on real-world transit use cases. GTFS-realtime v2.0 was unanimously approved by the GTFS community in August 2017. The research team also contributed to the development of the GTFS Best Practices, as well as several other improvements to the GTFS and GTFS-realtime governance and specifications. An open-source GTFS-realtime Validator tool was developed, which transit agencies and AVL vendors can use to quickly identify problems in GTFS-realtime feeds. The research team also developed another tool, the Transit Feed Quality Calculator, which can quickly validate a large number of transit feeds using the GTFS-realtime Validator as a software library. This tool illustrates the challenges that the industry currently faces in the area of GTFS-realtime feed quality control, with 54 of the 78 evaluated feeds having a number of integrity errors in a single GTFS-realtime message. It is very likely that continued monitoring of these feeds over multiple messages would discover even more errors.

Going forward, the additional clarity in GTFS-realtime v2.0 and GTFS Best Practices should help feed producers provide better quality data, which should help drive this error count down. Additionally, the GTFS-realtime Validator should be easier for transit agencies and AVL vendors to identify problems in a feed so they can spend less time hunting for examples of errors and more time fixing the underlying problem.

6.1 FUTURE WORK

Now that GTFS-realtime v2.0 has been accepted by the community, future work should focus on encouraging transit agencies to adopt it, especially when they create Requests for Proposals (RFPs) for new AVL systems. The research team has created a blog post “What’s new in GTFS-realtime v2.0” [14] that can be shared with interested parties, and has presented GTFS-realtime v2.0 at a TransitCenter Transit Data Workshop [12] as well as a paper and poster at the 2018 Transportation Research Board conference [13]. Future conference presentations and webinars can utilize these same materials. Given that the U.S. Department of Transportation has already requested GTFS data from agencies as part of the National Transit Map initiative [15], transit
agencies may benefit from similar guidance regarding GTFS-realtime emerging as a de facto standard.

During the development of the GTFS-realtime Validator software, the research team discovered additional issues that require clarification via new proposals to the GTFS community, either by the research team or by others [16]. Future work should include collaborating with the GTFS community to write proposals to address these gray areas in the specification. Potential new rules (identified with the “new rule” label [17]) could also be added to the GTFS-realtime Validator, including a few that first require clarification in the GTFS-realtime specification. Areas requiring clarification are documented on GitHub [16] and include:

- Should stop_time_updates be propagated across trips in the same block?
- Are delays propagated downstream passed SKIPPED stops?
- Are stop_time_update.SKIPPED values propagated downstream?
- Should early arrivals/departures be propagated across timepoints?
- How to represent an alert that has no effect?
- What should consumer behavior for stop_time_updates with NO_DATA be?
- Do arrival and departure times need to be included in all stop_time_updates?
- Must stop_time_update stop and stop_sequence pairing match GTFS data stop_times.txt?

Several transit agencies voiced the desire to have a hosted instance of the GTFS-realtime Validator tool instead of running it themselves, largely due to internal agency IT restrictions that prevent the installation of new software on their computers. Future work should focus on establishing a long-term server where the GTFS-realtime Validator tool can be hosted – some preliminary work to achieve this goal has been accomplished and can be extended to support this task in the future [18].

Discussions with some transit agencies also uncovered a lack of knowledge for what is currently possible with GTFS-realtime. For example, some agencies were unaware that you could announce canceled service (e.g., trips) for a day and route in the feed. Future work could focus on creating better documentation in non-technical language that identifies these common use cases and clearly outlines what is and is not possible given the current specification.

Future work could also focus on creating official GTFS-realtime Best Practices documentation via the community process, similar to how the GTFS Best Practices were established. For example, many of the warnings created for the GTFS-realtime Validator are likely to be considered best practices by the majority of the community, and could be officially adopted via a vote.

The Transit Feed Quality Calculator tool can be improved. Future work could focus on compiling additional feeds not currently documented in TransitFeeds.com, as well as utilizing a
CSV file to store some URLs for feeds that require API keys. Some preliminary work to collect data on which feeds require API keys has been done, and future work can build on these efforts [19].

Several agencies and application developers have expressed interest in a real-time “Data Dashboard” for GTFS-realtime feeds that would show the number of errors and warnings for all feeds for all agencies, as identified by the GTFS-realtime Validator. This website could be based on the hosted instance of the validator as discussed earlier or maintained as a separate service. Some have speculated that this would help personnel at all levels within the agency (from management to IT support) better understand how their agency is currently performing in the area of real-time data quality, and how they compare to their peers.

While the GTFS community has benefitted from the grassroots approach to governance in the past, given the substantial number of stakeholders for both GTFS and GTFS-realtime data the community may benefit from a more formal, organized structure going forward. The GTFS Best Practices effort showcases what can be accomplished relatively quickly by a smaller subset of organizations focused on accomplishing a specific goal, and a similar strategy will likely be required for creating GTFS-realtime Best Practices. This emerging coordination effort should be careful not to lose sight of the benefits of the grassroots approach that has made GTFS a widely adopted format (e.g., requiring producers and consumers for every new proposal to discourage speculative additions to the format). A more formal organization facilitating this coordination could also host useful resources and tools for producers and consumers of the data (e.g., the hosted GTFS-realtime Validator, the “Data Dashboard”).

Finally, there is research to be performed in the area of transit agency management and supporting information technology related to institutional barriers of acknowledging and resolving problems. For example, if a transit agency is provided with information to help them produce better quality transit data, will they take the steps to fix the problem? Would a hosted GTFS-realtime Validator or “Data Dashboard” help an agency improve their systems? The GTFS-realtime Validator tool can help produce quantitative metrics that could be tracked over time to try to answer some of these questions.
7.0 REFERENCES


realtime-validator/issues?q=is%3Aissue+is%3Aopen+label%3A%22GTFS%28-rt%29+spec+clarification%22

[17] University of South Florida. "GTFS-realtime - Issues that could result in new validation rules." Accessed February 20, 2018 from https://github.com/CUTR-at-USF/gtfs-realtime-validator/issues?q=is%3Aissue+is%3Aopen+label%3A%22new+rule%22


