Publishing and archiving planned and live public transport events with the Linked Connections framework

David Chaves-Fraga [a], Julian Rojas [b], Pieter Colpaert [b], Oscar Corcho [a] and Ruben Verborgh [b]
[a] *Ontology Engineering Group, Universidad Politécnica de Madrid, Spain*
[b] *IDLab, Department of Electronics and Information Systems, Ghent University-imec, Belgium*
E-mails: dchaves@fi.upm.es, julianandres.rojasmelendez@ugent.be, pieter.colpaert@ugent.be, ocorcho@fi.upm.es, ruben.verborgh@ugent.be

Abstract. Using Linked Data based approaches, companies and institutions are seeking ways to automate the adoption of Open Datasets. In the transport domain, data about planned events, live updates and historical data have to coexist to provide reliable data to route planning assistants. Linked Connections (LC) introduces a preliminary specification that allows cost-efficient publishing of the raw public transport data in linked information resources. This paper gives an overview of Linked Connections so far and supports claims with existing and novel experiments. Furthermore, (i) an extension of the current Linked Connections specification providing methods and vocabulary to deal with live data is provided; (ii) a Linked Connections Live server is developed that is able to process GTFS-RT feeds providing consistent identifiers; and (iii) an efficient management of historical data taking into account the size of each fragments exposed on the Web is described. We discover that the size of the fragments has a relevant impact on the performance of query evaluation. Based on our experiments conducted in 2018, an ideal Linked Connections fragment – for the use case of route planning with the current client – weighs about 500kb. This research scratches the surface on a Web ecosystem for route planning. In future works, we envision to find optimal fragmentation strategies of larger public transit networks for automated federated route planning.

Keywords: Linked Connections, Historical Data, Real time data, Reliable data, Linked Data Fragments

1. Introduction

In the current state of the Web of Data, a wide amount of that data are exposed following the principles of Linked Data[1]. Giving unique identifiers to each resource, representing the data using a shared and common vocabulary of a domain or the possibility of dereferencing each URI are some of the relevant aspects that made Linked Data as one the most common approaches to organize and expose the data on the Web[2]. This features allow third parties to query the data in a standard way, using for example, the corresponding query language for RDF, SPARQL[3], and the possibility of doing federation across multiple datasets[4]. However, today, a lot of domains need to deal with live and historical data where is important to maintain stable identifiers that remain valid over time. The problems regarding the manage of these types of data and the relation with Linked Data have not widely researched. One of the most relevant domains where contributions will have a relevant impact is the transport domain, where a complex environment with multiple types of data sources have to be managed to provide reliable data to information systems.

Since May 2017, one of the main motivations for developing solutions about multimodal and integrated travel information services is the publication of the new directive by the EU Commission about discoverability and access to public transport data across Europe. This document proposes the making of public transport data from providers available on national or common access points saved on databases, data warehouse or repositories. All the states will provide access to a unique common point following different static

```
{
  "@id": "http://madrid.linkedconnections.org/train/connections/1528278000000par_5_435_I11-5_I11_10%3A52%3A00_2_105_5__C4___",
  "@type": "Connection",
  departureStop: "http://madrid.linkedconnections.org/train/stops/par_5_43",
  arrivalStop: "http://madrid.linkedconnections.org/train/stops/par_5_34",
  departureTime: "2018-06-06T09:40:00.000Z",
  arrivalTime: "2018-06-06T09:42:00.000Z",
  "gtfs:trip": "http://madrid.linkedconnections.org/train/trips/5_I11-5_I11_10%3A52%3A00_2_105_5__C4___",
  "gtfs:route": "http://madrid.linkedconnections.org/train/routes/5__C4___",
  "gtfs:pickupType": "gtfs:Regular"
},
```

Fig. 1. Example of a connection at LC in JSON-LD

standards as Transmodel[1], Datex II[2] or GTFS[3] and real-time standards GTFS-RT[4] or SIRI[5]. So the domain requires solutions able to provide reliable data and to deal with the heterogeneity of access points and the data formats.

One of the main challenges when the Linked Data approaches deal with transport domain, where live and historical data has to be taken into account for providing consistent data to the information services, is how to ensure that the identifiers of each resource is stable and valid over the time. For example, if we define the connection entity as a departure-arrival pair, in previous works on Linked Connections[5], the URI of each connection was defined getting information from an static GTFS datasets. At the moment that live data is involved, that URIs are not consistents because the live information has to be taken into account. An example of the conceptualization of a connection is shown in the Figure 1. Other relevant challenge is how to manage the exposed historical data on the Web to allow an optimal query performance. One of the requirements of most relevant route planning algorithms is that the data have to be sorted by the time. In previous works of Linked Connections[6], we developed a server that paginates the list of connections in departure time intervals (10 minutes) and publishes these pages over HTTP. We have noticed that the clients were able to analyze the historical data but the performance was too low.

Our work is focused on providing an improvement of the current state of Linked Connections framework by allowing an efficient management of historical and live data with a standard vocabulary for the transport domain. This is relevant because two main reasons: (i) currently, a lot of transport companies are starting to provide access to their live services, but they the most common way to do it, is to develop an ad-hoc solutions like APIs or web services that only works locally [7] and (ii) the heterogeneity of the domain in terms of data formats will be a very relevant problem next years, especially in Europe, based on the proposal of the EU Commission so a standard framework will be needed.

In this paper we present the Linked Connections framework to provide reliable access to live and historical data. Our main contribution is the extension of previous version of the LC server, by providing an efficient management of live and historical data. First, we develop a library that gets information from static GTFS datasets and GTFS-RT data streams and exploit and integrate them following the LC vocabulary. Second, we modify the approach of Linked Connections for splitting the fragments of the data using a specific size of each fragment instead of the time. Third, we program a route planning algorithm in the top of the LC server to test if our approach is able to provide reliable data. Finally, we evaluate the improvements comparing the new version of the LC framework with our previous approaches, as base line, and we analyse the impact of the fragment size in the performance of a route plan.

The paper is organized as follows: Section 2 presents some of the related work about relevant approaches about exposing data on the web efficiently, previous steps about the Linked Connections framework, a description of the *de-facto* standard GTFS and the specification of relevant route planning algorithms. Section 3 describes our proposal about the improvements of the Linked Connections framework. Section 4 presents the design of our experiments. Section 5 describes the results we obtained evaluating our main contributions. Section 6 provides a brief discussion about the relevance of our contributions, and Section 7 presents conclusions and areas for future work.

2. Related Work

On the current state of the Web, huge amount of data are exposed following the principles of Linked Data. In this section, we describe the main contributions on this topic focused on an efficient exposition and on the transport domain, a description of previous approaches developed using the Linked Connections framework and the analysis of the model for transport data, GTFS, that supports our work. Finally we analyse the most relevant algorithms for route planning.

One of the most well-known alternatives to publish data on the Web is Linked Data [1]. Linked Data allows to identify in an unique way resources on the

[1] http://www.transmodel-cen.eu
[2] http://www.datex2.eu
[3] https://developers.google.com/transit/gtfs
[4] https://developers.google.com/transit/gtfs-realtime/
[5] http://www.transmodel-cen.eu/standards/siri/

Web using identifiers, or HTTP URIs. It is a method to distribute and scale data over large organizations such as the Web. When looking up this identifier by using the HTTP protocol or using a Web browser, a definition must be returned, including links towards potential other interesting resources, a practice called *dereferencing*. The triple format to be used in combination with URIs is standardized within RDF. The URIs used for these triples already existed in other data sources, and we thus favoured using the same identifiers. It is up to a data publisher to make a choice on which data sources can provide the identifiers for a certain type of entities.

A common problem in Linked Data is the availability of the triple stores. They provide a way to getting data using the SPARQL query language but at the moment of queries involved long periods of time, these approaches are not efficient[8]. The Linked Data Fragments(LDF)[9, 10] solve this issue fragmenting the data in several HTTP documents. Following this approach the store moves the load from server side to client side improving its availability. Comunica[11] is a framework that extends the possibilities of LDF, allowing to query other semantic interfaces as common SPARQL endpoint, RDF data dumps or HDT datasets[12].

Linked Connections[5] applies this approach to develop a cost-efficient solution based on a HTTP interface for transport data. The main assumption of LC is that the relevant data for route planners can be based on the connection concept. Basically, as the LC vocabulary[6] defines it, a connection describes a departure at a certain stop and an arrival at a different stop with their corresponding departure and arrival times and without any intermediate stop. A basic implementation of LC is shown in Figure 2, where the route planning algorithms have to analyse the connections (small rectangles) through the pages (big rectangles) and across the time to find the expected route. The join between the connections is possible because same resources have same identifiers, based on one of the principles of Linked Data. The Hydra Ontology[13] is used to specify the next and previous page links as well as how the resource itself should be discovered

It also relevant to describe the previous works that has been carried out using the specification of Linked Connections. For example, [14] describes a analyse the behaviour of a basic transport API and the Linked Connections framework for public transit route planning, comparing the CPU and query execution time. The authors found that, at the expense of a higher bandwidth consumption, more queries can be answered using LC than the origin-destination API. In [15] studies the impact of taking into account user preferences in a public transit route planning adding that features both on server and client and comparing the two solution on query execution time, cache performance and CPU usage on both sides. A first step for providing reliable access to historical and live data using Linked Connections is described in [6], where a mechanism is introduced to tackle the problem of the management of data modifications when live data is involved in route planning queries. Tripscore[7], a Linked Data client that consume several Linked Connections servers with live and historical data is also described in [16], where the Connection Scan Algorithm (CSA) is implemented as the route planning algorithm in top of the client[17]. Tripscore add multiple user preferences at the client side to provide an score for each possible route.

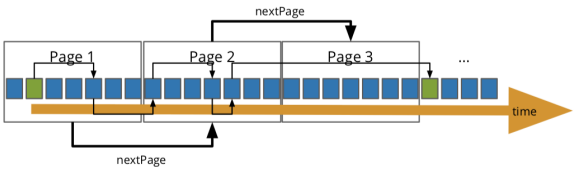

Fig. 2. Linked Connections implementation

All the solutions that we aforementioned are rely on the *de-facto* standard for representing public transport data, the General Transit Feed Specification or GTFS. This model, and its extension for real-time (GTFS-RT) is used by Google Maps[8] since 2005 but also by other route planners like Open Trip Planner [9] or Navita.io[10]. It is also the most common model used by the transport companies to expose their data on open data portals, like for example the Consorcio General de Transportes de Madrid[11], the TRAM in Barcelona[12] or the Belgium National Train System (SNCB). GTFS defines the headers of 13 types of CSV files and a set of rules the must be take into account when the dataset is created. Each file, as well as their headers, can be mandatory or optional and the have relations among

[6] http://semweb.mmlab.be/ns/linkedconnections

[7] www.tripscore.eu
[8] http://maps.google.es
[9] http://www.opentripplanner.org
[10] https://www.navitia.io
[11] http://datos.crtm.es
[12] https://opendata.tram.cat/

them as show in Figure 3. Linked Connections is getting the necessary information from a subset of the full dataset:

- stops: Individual locations where vehicles pick up or drop off passengers.
- calendar: Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.
- calendar_dates: Exceptions for the service IDs defined in the calendar file.
- stop_times: Times that a vehicle arrives at and departs from individual stops for each trip.
- trips: Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
- routes: Transit routes. A route is a group of trips that are displayed to riders as a single service.
- transfers: Rules for making connections at transfer points between routes.

In order to link the terms and identifiers define in these files with the Linked Open Data cloud, we used the Linked GTFS[13] vocabulary. We create mappings able to transform GTFS files to Linked GTFS following the CSV2RDF[18] W3C recommendation[14] but also using other standard OBDA mapping languages that are able to deal with CSV files[15], like RML[19] or R2RML[20].

The extension of GTFS for real-time, GTFS-RT[16] is a feed specification that allows public transport agencies to provide realtime updates about their fleet. The specification supports three types of information: (i) trips updates like delays, cancellations or change routes, (ii) service alerts like stop moved, unforeseen events affecting stations, routes, etc and (iii) information of vehicle positions including location and congestion level. The data exchange format is based on Protocol Buffers[17].

It is also important to mention the works about route planning algorithms, that can be developed on the top of the Linked Connections framework. The problem that these algorithms has to solve using the data is the Earliest Arrival Time (EAT). An EAT query consists of a departure stop, a departure time and a destination stop. The main goal is to find the fastest journey

[13] http://vocab.gtfs.org/terms
[14] https://github.com/OpenTransport/gtfs-csv2rdf
[15] https://github.com/dachafra/gtfsmappings
[16] https://developers.google.com/transit/gtfs-realtime/
[17] https://developers.google.com/protocol-buffers/

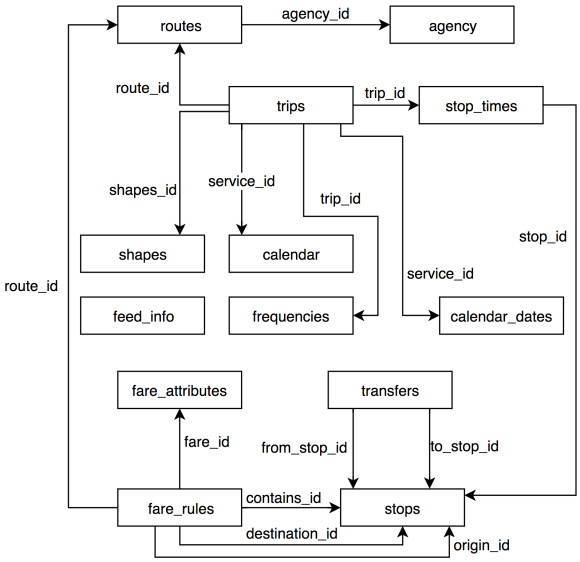

Fig. 3. The GTFS model and its primary relations

to the destination stop starting from the departure stop at the departure time. There are more complex route planning questions like the Minimum Expected Arrival Time (MEAT)[17] or *multi-criteria profile* queries[21–23]. The Connection Scan Algorithm (CSA) [17] is an approach that models the timetable data as a directed acyclic graph [24] using a stream of connections. As we defined before, a connection is a combination of a *departure stop* ($c_{depstop}$) with a *departure time* ($c_{deptime}$) and an *arrival stop* ($c_{arrstop}$) with an *arrival time* ($c_{arrtime}$). All the connections are combined in a stream of connections, sorted by increasing departure time. Thanks to this feature, it is sufficient to only consider the connections where the $c_{deptime}$ is later than the desirable departure time. The CSA algorithm works as follows, when a new scanned connection leads to a faster route to the arrival stop, the minimum spanning tree (MST) will be updated. This is the case when $c_{arrtime}$ is earlier than the actual EAT at $c_{arrstop}$, Finally the algorithm ends when the destination stop is added to the MST and the resulting journey can be obtained by following the path in the MST backwards.

In summary, after some proofs of concepts developed taking into account live and historical data in the LC framework and the current advances in the state of the work exposing the data on the Web efficiently, we think that it is the moment to improve the features of the current LC framework to create a standard mechanism for publish reliable public transport data. The main motivations to do that are, on one hand, the necessity to improve the current access of the transport

information services to the data, where at the moment that real-time is involved, the solutions are basically ad-hoc and they are not feasible if we are based on the proposal of the EU Commission for publish public transport data. On the other hand, supported by the characteristics of the transport data, where a complex data management environment emerges, the new approach that we present in this paper can serve as a source of inspiration for historical and live Linked Data management on the Web in other domains.

3. The Linked Connections Framework

In this section, we describe the Linked Connections framework for providing reliable access to live and historical data. First, we describe a summary of the Linked Connections specification including new properties about features of live data. Second, we describe the extensions we develop to deal with these types of data: the linked connections library for transforming live feeds into connections and the LC server for managing and exposing that connections on the Web efficiently.

3.1. The Linked Connections specification

The LC specification[18] explains the how to implement a data publishing sever, and explains what you can rely on when writing a route planning client. Following the Linked Data principles and the REST constraints, we make sure that from any HTTP response, hypermedia controls can be followed to discover the rest of the dataset. A "Linked Connections graph" is a paged collection of connections, describing the time transit vehicles leave and arrive. The Linked Connections vocabulary[19] defines the basic properties to used with the `lc:Connection` class:

- `lc:departureTime` It is a date-time, including delay, at which the vehicle will leave for the `lc:arrivalStop`.
- `lc:departureStop` The departure stop URI.
- `lc:departureDelay` Provides time in seconds when the `lc:departureTime` is not the planned departureTime.
- `lc:arrivalTime` It is a date-time, including delay, at which the vehicle will arrives at `lc:arrivalStop`.
- `lc:arrivalStop` The arrival stop URI.
- `lc:arrivalDelay` Provides time in second when the `lc:arrivalTime` is not the planned arrivalTime.

We also reuse the terms from the Linked GTFS vocabulary[20] in order to describe other properties of a `lc:Connection` class. This vocabulary it is aligned with the *de-facto* standard to exchange public transport data on the Web, GTFS:

- `gtfs:trip` Must be set to link a a `gtfs:Trip` with a `lc:Connection`, identifying whether another connection is part of the current trip of a vehicle.
- `gtfs:pickupType` Should be set to indicate whether people can be picked up at this stop. The possible values: `gtfs:Regular`, `gtfs:NotAvailable`, `gtfs:MustPhone` and `gtfs:MustCoordinateWithDriver`.
- `gtfs:dropOffType` Should be set to indicate whether people can be dropped off at this stop. The possible values are the same as the defined in the `gtfs:pickupType` property.

It is important to remark that each Linked Connections page must contain some metadata about itself. The URL after redirection, or the one indicated by the Location HTTP header, therefore must occur in the triples of the response. The current page must contain the hypermedia controls to discover under what conditions the data can be legally reused, and must contain the hypermedia controls to understand how to navigate through the paged collection(s). A different ways exist to implement the paging strategy. At least one of the strategies must be implemented for a Linked Connections client to fin the next pages to precessed:

1. Each response must contain a `hydra:next` and `hydra:previous` page link.
2. Each response should contain a `hydra:search` describing that you can search for a specific time, and that the client will be redirected to a page containing information about that timestamp. To describe this functionality the `hydra:property` `lc:departureTimeQuery` is used. An example is shown in the Figure 4

Finally, for each document that is published by a Linked Connections server, a Cross Origin Resource

[18] https://linkedconnections.org/specification/
[19] http://semweb.mmlab.be/ns/linkedconnections#
[20] http://vocab.gtfs.org/terms

```
{
  "@id": "https://graph.irail.be/sncb/connections?departureTime=2017-12-22T14:00:00.000Z",
  "@type": "hydra:PagedCollection",
  "hydra:next": "https://graph.irail.be/sncb/connections?departureTime=2017-12-22T14:10:00.000Z",
  "hydra:previous": "https://graph.irail.be/sncb/connections?departureTime=2017-12-22T13:50:00.000Z",
  "hydra:search": {
    "@type": "hydra:IriTemplate",
    "hydra:template": "https://graph.irail.be/sncb/connections/{?departureTime}",
    "hydra:variableRepresentation": "hydra:BasicRepresentation",
    "hydra:mapping": {
      "@type": "IriTemplateMapping",
      "hydra:variable": "departureTime",
      "hydra:required": true,
      "hydra:property": "lc:departureTimeQuery"
    }
  }
}
```

Fig. 4. Metadata LC response

Sharing[21] HTTP header must set the property Access-Control-Allow-Origin for sharing the response with any origin. The server also should implement thorough caching strategies, as the cachability is one of the biggest advantages of the Linked Connections framework. Both conditional requests[22] as regular caching[23] are recommended. As every connection will need to have a unique an persistent identifier, an HTTP URI, the server must support at least one RDF1.1 format, such as JSON-LD[25], TriG[26] or N-Quads[27]. The connection URI should follow the Linked Data principles[1].

3.2. Live data in Linked Connections

As we aforementioned, the transport domain should involve in their route planning algorithms the live data to provide consistent information to the passengers and improve the current informational services. The *defacto* standard for exchange transport data on the Web, GTFS, has created a version for live updates, GTFS-RT. Providing globally unique identifiers to the different entities that comprise a public transport network is fundamental to lower the adoption cost of public transport data in route-planning applications. Specifically in the case of live updates about the schedules is important to maintain stable identifiers that remain valid over time. Here we use the Linked Data principles to transform schedule updates given in the GTFS-RT format to Linked Connections and we give the option to serialize them in JSON, CSV or RDF (turtle, N-Triples or JSON-LD) format.

The URI strategy to be used during the conversion process is given following the RFC 6570[24] specification for URI templates. The parameters used to build the URIs are given following an object-like notation (object.variable) where the left side references a CSV file present in the provided GTFS data source and the right side references a specific column of such file. We use the data from a reference GTFS data source to create the URIs as with only the data present in a GTFS-RT update may not be feasible to create persistent URIs. As for the variables, any column that exists in those files can be referenced. The GFTS files that can be used to create the URIs are routes and trips files. As for the variables, any column that exists in those files can be referenced. A simple example is shown in Figure 5 and an standard template is also available[25]. Next we describe how are the entities URIs build based on these templates :

- **stop**: A Linked Connection references two different stops (departure and arrival stop). The data used to build these specific URIs comes directly from the GTFS-RT update, so we do not specify any CSV file and header from the reference GTFS data source. The variable name chosen in our case is the stop_id but it can be freely named.
- **route**: For the route identifier we rely on the routes.route_short_name and the trips.trip_short_name variables.
- **connection**: For a connection identifier we resort to its departure stop with connection.departureStop, its departure time with connection.departureTime(YYYYMMDD), and the routes.route_short_name and the trips.trip_short_name. In this case we reference a special entity we called connection which contains the related basic data that can be extracted from a GTFS-RT update for every Linked Connection. A connection entity contains these parameters that can be used on the URIs definition: connection.departureStop, connection.arrivalStop, connection.departureTime and connection.arrivalTime. As both departureTime and arrivalTime are date objects, the expected format can be defined using brackets.
- **trip**: In the case of the trip we add the expected departure time on top of the route URI based on the associated connection.departureTime(YYYYMMDD) and the information about the delay.

[21] http://enable-cors.org/

[22] https://developer.mozilla.org/en-US/docs/Web/HTTP/Conditional_requests

[23] https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching

[24] https://tools.ietf.org/html/rfc6570

[25] https://github.com/linkedconnections/gtfsrt2lc/blob/master/uris_template_example.json

```
{
  "@id": "http://example.org/connections/8861200/20180608/IC2110",
  "@type": "Connection",
  "departureStop": "http://example.org/stations/8861200",
  "arrivalStop": "http://example.org/stations/8861416",
  "departureTime": "2018-05-23T09:32:00.000Z",
  "arrivalTime": "2018-05-23T09:35:00.000Z",
  "departureDelay": 60,
  "arrivalDelay": 0,
  "direction": "Luxembourg (l)",
  "gtfs:trip": "http://example.org/trips/IC2110/20180608",
  "gtfs:route": "http://example.org/routes/IC2110"
}
```


Fig. 5. Connection example with live updates

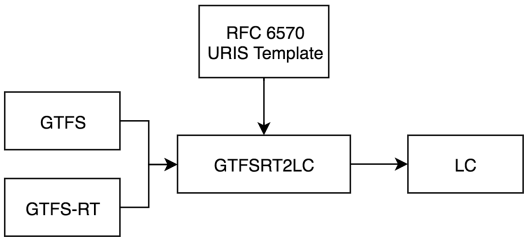

Fig. 6. The GTFSRT2LC tool

Finally, we developed a tool[26] that analyses the information from an static GTFS dataset and the updates from a GTFS-RT feed, exploits the implicit relations among the CSV files and creates the live Linked Connections feed in the desirable format with the help of the URIs template, as it is shown in Figure 6.

3.3. Managing live and historical data with the Linked Connections server

The third main contribution of this paper, after develop a way to create Linked Connections feeds taking into account live data, is how to provide reliable access to historical and live transport data in an cost-efficiently way. For that reason, we develop a Linked Connections server which exposes data fragments using JSON-LD serialization format. First, the server uses the GTFSRT2LC library for creating the Linked Connections feed as we describe in the above section. After that, the fragmenting process starts using the information provided in the configuration of the server about the fragment size and the fragment time span.

As we aforementioned, the previous process for fragmenting LC are based on the time, using this variable to pagination the data and creating an heterogeneous environment in terms of the fragment size. It is the main reason why we start to fragmenting the historical data of the LC feed homogeneously providing a specific fragment size, that has a relevant effect in terms of performance, as we demonstrate in the Section 5. We also fragment the LC live feed using a time span.

[26]https://github.com/linkedconnections/gtfsrt2lc

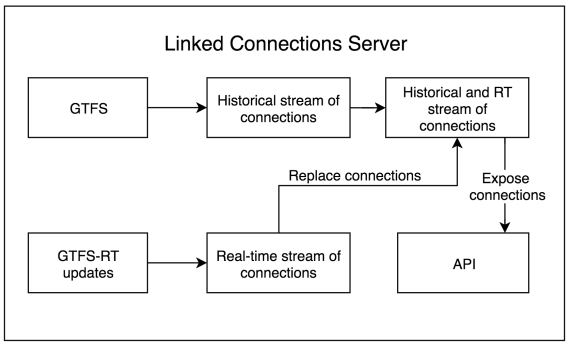

Fig. 7. The Linked Connections server

The server is able to manage both, the historical and the live connections. First, the server create the Linked Connections from the information provided by a GTFS dataset and fragment the historical data into several fragments. After that, if the transport system has an open live service, the sever starts to process the updates from the GTFS-RT feed and create another stream of connections based on that information. After that, the server searches the correspondence between the historical and live connections based on the URIs of the trips and replace the historical connections with the ones that contains live updates. Finally, the server exposes the stream of historical and live connections on an API following the fragmentation approach. This process, shown in Figure 7, is repeated every time that the server process an update.

The server also allows querying historic data by means of the Memento Framework[28] which enables time-based content negotiation over HTTP. By using the Accept-Datetime header a client can request the state of a resource at a given moment. If existing, the server will respond with a 302 Found containing the URI of the stored version of such resource. So if a request is made to obtain a Linked Connections fragment identified by a departure time, the headers of the fragment will provide the information about the Accept-Datetime. That means that is possible to know what is the state of the delays at a given date-time for a set of connections which guarantees the reliable of the data.

4. Evaluation Design

We design a set of experiments in order to evaluate our approach. In this section we describe the main features of our developments, data used for the experiments and the testbed we created. We implement different components in JavaScript for the Node.js platform. We chose JavaScript as it allows us to use the components both on a command-line environment as well as in the browser.

- GTFSRT2LC: A tool to convert live updates and timetables as open data to Linked Connections Vocabulary.
- LC server: Publishes streams of connections in JSON-LD, as we explain in the Section 3.
- CSA algorithm: We developed the CSA algorithm on the top of the Linked Connections Framework as the client.

We also use other tools that we developed previously, as the GTFS2LC[27] library, that it is able to convert static timetables as open data to Linked Connections Vocabulary and the Memento Protocol on the top of the Linked Connections Server[6], to enable reliable access to historical data, as we describe in the Section 3.

These tools are combined in different set-ups with a route planning algorithm in the client-side:

- Historical data fragmentation by size without cache: The first experiment executes the queries taking into account only historical data where the size of the exposed fragments are always the same. Client caching is disabled, making this simulate the LC without cache set-up, where every request could originate from an end-user's device. We test different fragments size to test how it affects to the performance.
- Historical data fragmentation by size with cache: The second experiment does the same as the first, except that it uses a client side cache, and simulates the LC with cache set-up.
- Live and historical data fragmentation by size without cache: The third experiment does the same as the first, but at this tame, live data is also take into account, modifying the fragments size due to the delays. We also test different fragments sizes.

Query	Dataset	# of Connections

Table 1
LC testbed

- Live and historical data fragmentation by size with cache: The fourth experiment does the same as the third except that it uses a client side cache, and simulates the LC with cache set-up.
- Live and historical data fragmentation by time without cache: The fifth experiment fragments the data based on the time, as our previous approaches with the client cache disable. We want to test if the performance of our pagination by the size is better than by the time.
- Live and historical data fragmentation by time with cache: The sixth does the same as the fifth but it uses the client side cache.

As far as we are aware of, there are no general-purpose benchmarks in the state of the art for testing our main contributions with the Linked Connections framework. Therefore, we have created a testbed as follows: 1) we have selected and downloaded 4 different representative GTFS datasets, available as open data, that also cover several geographical locations, 2) we have created several route planning queries (departure and arrival stop with a departure time) that reflect real-life trips, with multiple levels of complexity and 3) we have evaluated these queries in the different set ups take into account the features of each dataset.

The used GTFS datasets and GTFS-RT feeds are available as open data on the Web. There are two datasets that represent the Belgian Rail Transport System (NMBS) and the Flemish Transport Company (Delijn) and other two that represent the Tram Company of Barcelona (TBS and TBX). The first two one, also provide open access to their real-time following the standard GTFS-RT format but the second ones have developed an ad-hoc live API[28]. We are currently developing a tool to transform the ad-hoc live information to the GTFS-RT format but at this moment, only historical data can be taken into account in these cases.

We ran the experiments on a... Our experiments can be reproduced using the code at https://github.com/julianrojas87/linked-connections-server and the

[27] https://github.com/linkedconnections/gtfs2lc

[28] https://opendata.tram.cat/manual_en.pdf

```
"companyName": "NMBS",
"downloadUrl": "http://www.belgianrail.be/opendata/transit.zip"
"downloadOnLaunch": true,
"updatePeriod": "0 0 2 * * *",
"fragmentSize": 400000,
```

Fig. 8. NMBS GTFS configuration

```
"realTimeData": {
   "downloadUrl": "http://www.belgianrail.be/opendata/updates.bin",
   "updatePeriod": "*/30 * * * * *",
   "fragmentTimeSpan": 600,
   "compressionPeriod": "0 0 3 * * *"
}
```

Fig. 9. NMBS GTFS-RT configuration

testbed and the data at https://github.com/cef-oasis/linkedconnections-tests

5. A running example: Providing reliable access to NMBS data

The NMBS is the national company of trains in Belgium. They provide their static and live data as open data in GTFS and GTFS-RT formats. We use this case to explain how our approach works.

As we incorporate the GTFS2LC and GTFSRT2LC tools to the Linked Connections server, we only have to configure it to start producing the fragments of live and historical data. The basic configuration for the GTFS dataset is shown in Figure 8 where the companyName must be a unique identifier of the dataset, the downloadUrl must be a public URL where the server has to download the GTFS dataset, the downloadOnLanch boolean indicates if the dataset must be downloaded and processed upon server launch, the updatePeriod is a node-cron[29] expression to check if a new version of the dataset has been published for creating a new LC feed and finally, the fragmentSize represents the size of each fragment in bytes.

For the GTFS-RT feed the configuration is show in the Figure 9 where the downloadUrl is the URL where the updates are published, the updatePeriod is a node-cron expression that indicates how often should the server look for and process a new version of the feed, the fragmentTimeSpain defines the fragmentation of live data and it represents the time span of every fragment in second and finally the compressionPerior is also a node-cron expression that defines how often will the live data be compressed using gzip in order to reduce storage consumption.

Finally, we have to configure the URIs using the template we describe in the Section 3.2 and launch the server. After the GTFS datasets have been processed, we are able to start querying the data. All the fragments have a uniform starting point, which is the expected departure time of its first `lc:connection`. So the clients can start accessing the transport data using using the departure time as a parameter like this example: http://host:port/NMBS/connections?departureTime=2017-08-11T16:45:00.000Z. In this example, the client is the CSA route planning algorithm....

[29] https://www.npmjs.com/package/node-cron

6. Results

7. Discussion

8. Conclusions and Future work

{Pieter says: Please provide a conclusion here, not just a summary!} In this work we present the Linked Connections framework for providing reliable access to historical and live data. First, we analyse the previous works made over the Linked Connections and their problems on efficiency when historical and live data are taking into account. Second, we extends the Linked Connections vocabulary to consider these types of data in the vocabulary. Third, we develop a tool for creating stream of connections based on the information provided by live updates. Fourth, we adapt the LC server to efficiently manage and expose the transport data on the Web. Finally, we test our approach, analysing how the fragment size affects to the performance of the route planning algorithms and compare them with our previous approaches, that paginate the data by a time span. In the future work, we develop tools able to transform other transport format standards like Datex II, Transmodel or SIRI and expose streams of connections following LC on the Web. We also want to create a method able to find the optimal fragment size of a dataset based on several features like the type of the transport, the size of the full Linked Connections feed, the variability of the updates or the geographical location. Improving the discoverability of transport datasets is another line we want to take into account, we started to work on a registry by extending DCAT-AP to a Transport application pro-

file[30] to improve the discoverability of these datasets. Finally, we want to create a standard benchmarking with GTFS and GTFS-RT datasets, several representative route planning and ad-hoc relevant metrics.

Acknowledgements

This work has been partially supported by a predoctoral grant from the I+D+i program of the Universidad Politécnica de Madrid and also by the CEF European project OASIS CEF-26696297.

References

[1] C. Bizer, T. Heath and T. Berners-Lee, Linked data-the story so far, *International journal on semantic web and information systems* **5**(3) (2009), 1–22.

[2] T. Heath and C. Bizer, Linked data: Evolving the web into a global data space, *Synthesis lectures on the semantic web: theory and technology* **1**(1) (2011), 1–136.

[3] E. Prud, A. Seaborne et al., SPARQL query language for RDF (2006).

[4] C. Buil-Aranda, M. Arenas, O. Corcho and A. Polleres, Federating queries in SPARQL 1.1: Syntax, semantics and evaluation, *Web Semantics: Science, Services and Agents on the World Wide Web* **18**(1) (2013), 1–17.

[5] P. Colpaert, A. Llaves, R. Verborgh, O. Corcho, E. Mannens and R. Van de Walle, Intermodal public transit routing using Liked Connections, in: *International Semantic Web Conference: Posters and Demos*, 2015, pp. 1–5.

[6] J.A. Rojas Melendez, D. Chaves, P. Colpaert, R. Verborgh and E. Mannens, Providing reliable access to real-time and historic public transport data using linked v-connections, in: *ISWC2017, the 16e International Semantic Web Conference*, Vol. 1931, 2017, pp. 1–4.

[7] P. Colpaert, A. Chua, R. Verborgh, E. Mannens, R. Van de Walle and A. Vande Moere, What public transit API logs tell us about travel flows, in: *Proceedings of the 25th International Conference Companion on World Wide Web*, International World Wide Web Conferences Steering Committee, 2016, pp. 873–878.

[8] R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens and R. Van de Walle, Querying datasets on the web with high availability, in: *International Semantic Web Conference*, Springer, 2014, pp. 180–196.

[9] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck and P. Colpaert, Triple Pattern Fragments: a low-cost knowledge graph interface for the Web, *Web Semantics: Science, Services and Agents on the World Wide Web* **37** (2016), 184–206.

[10] R. Verborgh, M. Vander Sande, P. Colpaert, S. Coppens, E. Mannens and R. Van de Walle, Web-Scale Querying through Linked Data Fragments, in: *LDOW*, Citeseer, 2014.

[30] https://github.com/cef-oasis/DCAT-AP

[11] R. Taelman, J. Van Herwegen, M. Vander Sande and R. Verborgh, Comunica: a Modular SPARQL Query Engine for the Web, in: *International Semantic Web Conference*, 2018.

[12] J.D. Fernández, M.A. Martínez-Prieto, C. Gutiérrez, A. Polleres and M. Arias, Binary RDF representation for publication and exchange (HDT), *Web Semantics: Science, Services and Agents on the World Wide Web* **19** (2013), 22–41.

[13] M. Lanthaler and C. Gütl, Hydra: A Vocabulary for Hypermedia-Driven Web APIs., *LDOW* **996** (2013).

[14] P. Colpaert, R. Verborgh and E. Mannens, Public Transit Route Planning Through Lightweight Linked Data Interfaces, in: *International Conference on Web Engineering*, Springer, 2017, pp. 403–411.

[15] P. Colpaert, S. Ballieu, R. Verborgh and E. Mannens, The Impact of an Extra Feature on the Scalability of Linked Connections., in: *COLD@ ISWC*, 2016.

[16] D. Chaves-Fraga, J. Rojas, P.-J. Vandenberghe, P. Colpaert and O. Corcho, The tripscore Linked Data client: calculating specific summaries over large time series, in: *Proceedings of the Workshop on Decentralizing the Semantic Web (DeSemWeb)*, 2017.

[17] J. Dibbelt, T. Pajor, B. Strasser and D. Wagner, Intriguingly simple and fast transit routing, in: *International Symposium on Experimental Algorithms*, Springer, 2013, pp. 43–54.

[18] J. Tennison, G. Kellogg and I. Herman, Model for tabular data and metadata on the web. W3C recommendation, *World Wide Web Consortium (W3C)* (2015).

[19] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens and R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data., in: *LDOW*, 2014.

[20] S. Das, S. Sundara and R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012, *Cambridge, MA: World Wide Web Consortium (W3C)(www. w3. org/TR/r2rml)* (2012).

[21] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev and F. Viger, Fast routing in very large public transportation networks using transfer patterns, in: *European Symposium on Algorithms*, Springer, 2010, pp. 290–301.

[22] D. Delling, T. Pajor and R.F. Werneck, Round-based public transit routing, *Transportation Science* **49**(3) (2014), 591–604.

[23] S. Witt, Trip-based public transit routing, in: *Algorithms-ESA 2015*, Springer, 2015, pp. 1025–1036.

[24] B. Strasser and D. Wagner, Connection scan accelerated, in: *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, SIAM, 2014, pp. 125–137.

[25] W.W.W. Consortium et al., JSON-LD 1.0: a JSON-based serialization for linked data (2014).

[26] C. Bizer and R. Cyganiak, RDF 1.1 TriG. RDF Dataset Language, *W3C recommendation. W3C* **25** (2014).

[27] R. Cyganiak, A. Harth and A. Hogan, N-quads: Extending n-triples with context, *W3C Recommendation* (2008).

[28] H. Van de Sompel, R. Sanderson, M.L. Nelson, L.L. Balakireva, H. Shankar and S. Ainsworth, An HTTP-based versioning mechanism for linked data, *arXiv preprint arXiv:1003.3661* (2010).